



varian data
620/i
computer
manual



varian data machines

FEATURES OF THE DATA 620/ i SERIES COMPUTERS

Field Proven Software

Silicon Monolithic Integrated Circuits (DTL and TTL)

9 Hardware Registers

Over 100 Basic Commands

6 Addressing Modes

Direct Addressing to 2,048 or 32,768 Words

16- or 18-Bit Words

Expansion to 32,768 Words

Hardware Index Registers

Party Line I/O Facility

Micro-EXEC Option

10-1/2 Inches of Rack Space

Less than 70 Pounds (Mainframe and power supply)

340 Watts

NPN or PNP (Optional) I/O Levels

Interface Ease

Compatible with DATA 620 Computer

Plug-In Expandable

Low Cost

PREFACE

This manual is published to provide a complete description of the Varian DATA 620/i computer in a single document. It describes the hardware, the interfaces and the software, making it a useful reference for system designers, programmers and ultimate users. There are two major parts of this manual; the first for the systems-oriented reader, the latter for the programming-oriented reader—with an overlap. The last section contains appendices for quick reference material. The various sections are cross-referenced to simplify the reader's task of finding exactly what he's looking for.

For those interested in a quick overview of the Varian DATA 620/i, the introductory section should suffice. It contains a "thumbnail sketch" of the computer and its capabilities.

For further details on the computer and its interfaces, read sections 2 through 8. These describe the functional design of the computer and provide specific information on the means for connecting it to other equipment. Some of the options that may be added to the basic system are described in section 9.

Information on the basic software is provided in sections 10 through 12. For the programmer who wants to work with the basic computer capabilities, section 3 describes instruction operation, section 10 describes the assembly system, and section 11 presents the available subroutines and utility package. The last section on FORTRAN is for the use of the application-oriented programmer.

The appendices in the back of this manual contain quick reference material.

CONTENTS

<u>Section</u>	<u>Title</u>	<u>Page</u>
1	SUMMARY OF THE FEATURES OF THE VARIAN DATA 620/i	1
1.1	Introduction	1
1.2	The DATA 620/i Interfaces	1
1.3	Organization	2
1.4	Word Formats	4
1.5	Memory	5
1.6	Reliability and Maintainability	5
1.7	Programmed Input/Output	6
1.8	Peripheral Equipment	9
1.9	System Software	10
1.10	User Services	11
1.11	DATA 620/i Specifications	12
2	SYSTEM DESCRIPTION	15
2.1	Computer Organization.	15
2.2	Computer Word Formats.	17
3	OPERATIONAL INSTRUCTIONS	23
3.1	Single-Word Instructions	23
3.2	Double-Word Instructions	39
4	INPUT/OUTPUT OPERATION	57
4.1	Organization	57
4.2	Program Control Functions	58
4.3	Optional Automatic Control Functions	62
5	CONTROL CONSOLE OPERATION	65
5.1	Controls and Indicators	65
5.2	Manual Operation	65

CONTENTS (Continued)

<u>Section</u>	<u>Title</u>	<u>Page</u>
6	DATA 620/i INPUT/OUTPUT SYSTEM	71
6.1	General Description	71
6.2	Basic I/O Bus Signal Interface	72
6.3	I/O Command Operation	75
7	DIRECT-MEMORY-ACCESS-AND-INTERRUPT OPTION.	91
7.1	Direct-Memory-Access Function	91
7.2	Interrupt Function	92
7.3	Direct-Memory-Access-and-Interrupt Control Lines.	92
7.4	Details of Direct-Memory-Access Sequence	92
7.5	Details of Interrupt	93
7.6	External Device Priority	97
8	I/O CABLE INTERFACE CHARACTERISTICS.	101
8.1	I/O Bus Line Configurations	101
8.2	I/O Cable Driver and Receiver	101
8.3	I/O Cable and Connector	101
9	OPTIONS	105
9.1	Multiply, Divide, Extended Addressing	105
9.2	Memory/Peripheral Controller Expansion Chassis.	105
9.3	Memory Module	105
9.4	Direct Memory Access and Interrupt.	106
9.5	Real-Time Clock	106
9.6	Power Fail/Restart	106
9.7	Priority Interrupt.	106
9.8	Memory Protect System Module	107
9.9	Negative I/O	107
9.10	Micro-EXEC	107
10	DATA 620/i ASSEMBLY SYSTEM	111
10.1	Introduction	111
10.2	The DAS Source Language	111

CONTENTS (Continued)

<u>Section</u>	<u>Title</u>	<u>Page</u>
10.3	DATA 620/i Instructions	118
10.4	DAS Pseudo Instructions	123
10.5	DAS Output List	132
10.6	DAS Error Messages	133
10.7	Operating the DAS Assembly System	133
10.8	FORTRAN Pseudo Instructions.	134
11	DATA 620/i SUBROUTINES	139
11.1	Introduction	139
11.2	Programming Standards	139
11.3	Program Description.	140
11.4	Programmed Arithmetic.	142
11.5	Elementary Functions	175
11.6	Utility and Debugging Routines	206
12	FORTRAN	215
12.1	Basic FORTRAN Concepts	215
12.2	Data.	219
12.3	Specifications and Statements.	221
12.4	Expressions and Assignments	224
12.5	Control Statements	226
12.6	Input/Output Statements	231
12.7	Programs and Subprograms	239
12.8	FORTRAN Operating Instructions	243
APPENDICES		
A	DATA 620/i System Organization	249
B	Number Systems	251
C	DATA 620/i Instructions	261
D	I/O Control Codes and Device Addresses	281
E	Standard Character Codes	291
F	DAS Reference Information	297
G	Standard DATA 620/i Subroutines	301
H	FORTRAN Reference Information	303

SECTION 1

SUMMARY OF THE FEATURES OF THE VARIAN DATA 620/i

1.1 INTRODUCTION

DATA 620/i is a system-oriented digital computer, designed as a powerful system computer to fill the gap between special purpose digital hardware and general purpose computers. DATA 620/i meets all the requirements of a true system computer — powerful computing ability, easy interfacing, modular design and construction for expandability, integrated circuit reliability, low cost and compact size.

In addition, DATA 620/i offers a number of features simply not available on other computers — like Parly Line communication, quick and easy memory expandability from 4096 to 32,768 words of 16 or 18 bits, and a unique Micro-EXEC microstep sequencing technique. DATA 620/i comes with a complete set of field-proven software, developed and perfected on the DATA 620.

DATA 620/i has a bigger instruction set, 1/2 the components, and costs less than any computer in its class. This is why it so efficiently and economically solves system problems previously considered too difficult or expensive for computer solution.

DATA 620/i offers a wide variety of peripherals and options, allowing the user to select only those features specifically required for his application, and providing the optimum amount of computer power per dollar.

THE DATA 620/i

As a physical system component, DATA 620/i processors are compact in size, occupying only 10.5 inches of rack space. They are accessible from the front like other system components, and they are reliable and maintainable. The contents of five operational registers can be displayed on the front panel.

Eighty-five percent of the processor operation can be verified from the front panel without the use of an oscilloscope. As the controlling element in a system, a DATA 620/i has the "raw" data manipulating power of a much more costly computer. The instruction set includes over 100 basic machine commands. The register change command is micro-programmable with other 100 useful combinations. The processing characteristic can be adapted to specific requirements through an optional Micro-EXEC facility that permits software programs to be hardware implemented.

1.2 THE DATA 620/i INTERFACES

The DATA 620/i series was designed to not only provide the complete spectrum of interface capabilities required in a system computer, but to also allow the user to tailor the computer for his specific application. To attain this goal, all of the Input/Output features are offered as options. Among these facilities are: Direct Memory Access, Real Time Clock, Power Failure Protect and the Buffer Interlace Controller. These features, combined with priority interrupts, external sense lines, external control lines and the

proprietary Micro-EXEC technique give the DATA 620/i family virtually every I/O capability available.

THE DATA 620/i USER INTERFACE

As must be the case in any machine that is required to do—and do well—a large number of data manipulation tasks which are unspecified in advance, flexibility was the motif in designing the DATA 620/i software package. The goal was to achieve flexibility without creating big problems on the one hand, or falling into the easy habit of accepting hardware/software tradeoffs on the other hand. In the DATA 620/i, hardware and software features reinforce each other. For example, there are five modes of single-word addressing, one of which permits direct addressing of four times as many words in store as is normally possible with conventional designs. Multiply/divide instructions are available as options to meet more demanding computation speed requirements.

SYSTEM INTERFACE

The ability of the computer to adapt to the system is an excellent criterion for determining a true systems computer.

The design philosophy behind the 620/i Input/Output structure is not only to provide all of the capabilities needed in a system computer, but to allow the user to choose the particular capability needed for his particular application. The reasoning is: If the feature is needed, it can be provided as a low cost option; if the need is uncertain, it can be easily added in the field if and when it is needed.

The DATA 620/i family offers the widest range of interface facilities. These include Party Line Communication Bus, Multilevel Priority

Interrupts, External Sense Lines, External Control lines, Direct Memory Access and Interface Control.

1.3 ORGANIZATION

The DATA 620/i is organized with a unique bus structure, selection logic and nine registers. The organization provides universal internal information routing, buffered processing, micro-register change programming facility, information indexing without time penalty and the optional Direct Memory Access (cycle stealing) facility.

The organization optimizes the DATA 620/i for maximum I/O throughput, minimum elapsed time between successive input or output transfers, and minimum programming. Appendix A shows the functional organization of the busses and registers.

This unique organization makes possible the optional Micro-EXEC facility by which complex algorithms or additional instructions can be implemented with external hardware. The Micro-EXEC technique produces an increase in processing speed in excess of 500 percent over conventional stored program techniques. The bus structure of this computer family permits the system designer to overcome traditional barriers of processing speed, high-rate volume throughput and fixed mainframe characteristics. The four available busses are:

BUSSES

"L" bus provides a 12-bit parallel communications path from the L register to the address decoders in the memory modules.

"W" bus provides a parallel data communications path (16/18 bit) from the W registers to the memory module(s) (up to 8).

"C" bus provides the parallel path and selection logic for routing data between the arithmetic unit, the I/O unit, and the operational registers. This bus permits data to be uniquely or commonly transferred to the operational registers. It performs the distribution function for micro-programming, and provides a bi-directional parallel word path to the "Party Line". "C" bus is the central communication avenue and connects with all internal units of the processor. It is the key facility that permits Micro-EXEC to be implemented.

"S" bus provides the parallel path and selection logic for routing data between the operational registers and the arithmetic unit. It implements the select, gather and route function for micro-programming and Micro-EXEC.

Party Line I/O bus provides a 16/18-bit parallel bi-directional I/O communication path. This bus includes the control lines for transfer ready, sense, control, interrupt address and acknowledge and information entry. The "Party Line" is packaged as one-cable, and each peripheral device has a Party Line connector and a Party Line extender connector. The device and the Party Line form a "daisy chain" whereby additional I/O controllers can be added on site and on a plug-in basis.

REGISTERS

Nine registers are provided with a basic processor. Four of the nine registers are incorporated to provide buffering to satisfy real-time system requirements. All the arithmetic and control unit registers are multipurpose and can serve a unique micro-programming and Micro-EXEC function.

"A" register is a full-word register and is the high-order of the accumulator.

"B" register is a full-word register and is the low-order-half of the accumulator.

"X" register is a full-word register which permits indexing of memory addressing without adding time to accessing an indexed location.

"P" register is a full-word register and is the program counter.

"U" register is a full-word buffer which holds the instruction being executed.

"S" register is a 5-bit register which, in combination with the U register controls the length of shift instructions.

"L" register is the 16-bit memory location register.

"W" register is the memory word register and is full length (16 or 18 bits).

"R" register is a full-word buffer which holds the multiplicand or divisor in arithmetic operations.

MICRO EXEC

Micro-EXEC (Optional) is a technique by which the system designer has the option of externally combining and sequencing the processor's micro-steps to perform a complex macro-steps to perform a complex macro-function. Over 30 micro-step control lines are made available to the system user. These control functions are the micro-steps normally controlled by machine instructions.

They control memory, arithmetic unit, control unit, all registers, I/O and communication networks. The external control can operate the micro-steps as fast as five every 900 nanoseconds by utilizing the processor clock to

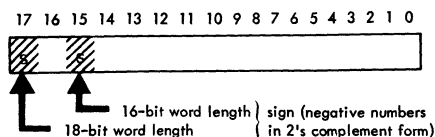
synchronize the micro-step operations. Micro-EXEC can be used to implement many types of algorithms. Typical functions are: convolutions, coordinate transformations, double precision arithmetic, table look ups, square root, limit checking, etc. Micro-Control can produce up to 10-to-1 speed advantage over stored programs and does not require core memory for the program. Opening new dimensions to the data system designer, Micro-EXEC makes practical an extremely fast processor with small or large memories. It permits the mode of processing to be controlled externally, and processing to be optimized for the system.

The processor organization and hardware provides the system engineer with the most flexibility available in off-the-shelf equipment. The standard options of Micro-EXEC, machine instructions, memory and I/O facilities provide functional adaptability and system optimization without engineering risk or unpredictable costs.

1.4 WORD FORMATS

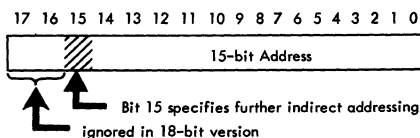
The DATA 620/i has three general categories of words: data, address and instruction. Each category has been optimized for the system environment. DATA 620/i processors and memories are available in either 16- or 18-bit word lengths (the 18-bit version is called the 622/i).

DATA FORMAT



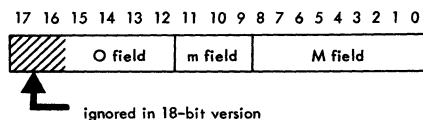
The details of how numbers are represented and arithmetic operations are performed in the 16-bit format are shown in Appendix B. For convenience in interpreting binary and octal values, conversion tables are also presented in Appendix B.

ADDRESS FORMAT



This address may be associated with an instruction as a double-word instruction, or it may be referred to by indirect addressing.

INSTRUCTION FORMAT



Details of the various sub-categories of instruction formats are presented in sections 2.2 and 3. In general there are two types: single-word and double-word. The single-word instruction is designed to enable the system user to write his programs in the minimum number of memory locations and have his program executed in minimum time. The double-word instruction has the same general format for the first word, and the second word is in either the data format ("Immediate" instructions) or the address format.

The instruction set is the most comprehensive available with "compact" computers or processors. The optional instruction sets have

specific value to certain applications and are available to refine the processors to those applications. The instruction set, variety, simplicity and power equates to economic optimization. The instruction list is presented in Appendix C.

1.5 MEMORY

The DATA 620/i uses general purpose random access ferrite magnetic core memories. They contain a proprietary thermal compensation technique which preserves the operating margins over the temperature range (5° to 45°C) without adjustment.

The memory communicates with the processor through a memory data bus and an address bus. Additional external (to mainframe) memory modules can be added simply by adding an optional memory adapter to the processor that permits the additional module to be "plugged in". The external memory module includes an adapter for the next memory module. The memory can be expanded to 32,768 words by the addition of 4K memory modules.

Memory cycle time is 1.8 μ sec; access time is 750 nanoseconds.

1.6 RELIABILITY AND MAINTAINABILITY

DTL and TTL integrated circuits are used throughout the DATA 620/i. These integrated circuits are general purpose digital logic, and are noted for low power consumption, high packing density, high noise rejection and reliability throughout the operating temperature range of 0° to 45°C. The low power equates to low heat generation and high reliability.

DATA 620/i's are produced under a quality control program designed and practiced to meet

MIL-Q-9858A, and to the intent of NPC 200-3. The mean-time-between-failures (MTBF) has been calculated for the basic processors to be over 7,500 hours. The mean-time-to-repair is estimated to be a few minutes.

DATA 620/i's are packaged to simplify maintenance. The integrated circuit board layout is unique using a "Bit Slice" layout. "Bit Slice" is a technique whereby all register and gating circuits associated with six bits are packaged on one card.

The structure is designed for easy access. All units of the processor are mounted to be easily removed to make all components and wiring easily accessible. The "big board" concept is used to permit easy trouble shooting.

Failure Detection: The source of faults in solid-state electronic equipment with conservative circuit and timing designs is from external causes. The external causes are power failures, power frequency failures, excessive heat and the failure of electro-mechanical peripheral devices. The DATA 620/i has been designed to prevent each of these fault sources from destroying the integrity of the system computer function.

- **Power Failure** — An optional Power Failure Protect System monitors power line voltage. If voltage is outside safe limits, a power fail interrupt is generated. The interrupt subroutine assures an orderly, safe shutdown. Upon restoration of power, the computer is automatically restarted at a designated memory location, and appropriate software provides an orderly restart.

- **Temperature** — A thermal sensor is embedded in the core memory to continually monitor internal temperature. If the temperature rises above the specified limit (45°C), the sensor produces a thermal alarm signal that is used to light the console alarm indicator and/or generate an interrupt line.
- **Operator Errors** — The control panel is electrically disconnected during "RUN" mode.
- **Memory Protect** — This option permits a top-priority executive, control, alarm, processing or monitor system to remain resident in memory while other programs are being processed.

These facilities provide the system engineer with the level of assurance needed to tackle the most demanding process control or real-time application where one failure can be extremely costly.

PHYSICAL

Packaging — The DATA 620/i family is packaged to offer the user maximum convenience, positioning, flexibility and space-saving economies. The memory, arithmetic and control unit, and the power supply and control console are three separate packages that, when connected, produce a compact unit that is 10-1/2 inches high, 22 inches deep and 19 inches wide. The compactness and light weight of the DATA 620/i series enables it to be used in facilities such as submarines, aircraft, etc.

Control Panel — The user-oriented design philosophy of the DATA 620/i console utilizes sound human engineering practices. The console has been developed to produce a pleasing image and still be functionally easy to use.

Proximity of related functions, minimum reflectivity, and other more subtle features such as length and distance of switches were used in the development of the console. The basic function of the console — to modify and monitor all operational registers — was achieved without a cluttering of switches that tend to confuse. A simple straight-forward instrument is the result.

ENVIRONMENTAL

The DATA 620/i connects to standard commercial single-phase 115 vac power. Power regulation is not required under normal commercial power conditions. Subflooring or conditioned air are not required. The DATA 620/i is equally at home in the shop, field, instrumentation room, classroom and laboratory.

1.7 PROGRAMMED INPUT/OUTPUT

The basic 620/i Processor is equipped with positive voltage level Party Line I/O bus. The Party Line is a bidirectional common communication channel containing the data and control lines required for system communication. Time-shared between the peripherals, it is designed to prevent conflicts or traffic jams under heavy communication loads. Each transmission contains the routing information as well as the data. It is transmitted as an entity which is not separable by interrupt. Thus, numerous devices can time share the Party Line. The transmission has two phases: The first phase is the route set-up, the second is the data transmission.

The Party Line permits plug-in expansion of all peripheral devices. The Party Line contains line drivers and line receivers to service up to ten peripheral devices. Each peripheral device contains a data buffer and Party Line adapter. Thus, no device can tie-up the Party

Line, and modifications to the computer are not required to add peripherals. Each device has a Party Line connector and a Party Line extender connector. The last device on the Party Line has a termination shoe on the extender connector. When another device is added, a Party Line cable is provided between the added and the last device. The termination shoe is moved to the added device.

The Party Line technique solves the troublesome problems usually encountered in time-shared operation and on-site system expansion.

The following interface features can be added to the basic Party Line.

DIRECT MEMORY ACCESS AND INTERRUPT LOGIC

This option provides direct memory access (cycle steal capability) from the Party Line I/O bus. With this feature, the user can design special system devices that cause the program to hesitate for 2.7 microseconds, during which time memory is accessed for data, if data is stored in memory. This trap operation bypasses the A, B, X and P registers, thus allowing the program to proceed normally. One interrupt level is provided with the option.

INTERRUPT SYSTEM

The DATA 620/i has a multilevel priority interrupt system with single-instruction execute, group enable/disable, and selective arm/disarm capability. Each interrupt line is assigned a unique memory destination address that is the first of a pair of locations. The system is modular and expandable in groups of eight levels, up to 64 levels.

The interrupt system is automatically scanned every 900 nanoseconds and the interrupt is recognized before the fetch cycle of the next

instruction to be executed. If signals exist on one or more interrupt lines, the highest priority is recognized. An interrupt functional response to an external device can be accomplished in as little as two memory cycles.

Each group of eight or sixteen interrupts can be enabled/disabled, and contains a 16-bit mask register that controls the individual interrupt lines. The program can maintain the hardware order of priority or reorder to meet dynamic queueing.

Acknowledgement of an interrupt by the Central Processor causes the instruction located at the memory destination address of the interrupt to be executed. The instruction can be any of the DATA 620/i instruction set. This technique permits the interrupts to be of the single-execute type, whereby single-instruction responses to external signals can be serviced in one instruction period. If the executed instruction is a jump and mark (JMPM), the interrupt system is automatically inhibited to permit the inhibit to be terminated under program control. The DATA 620/i interrupt system provides the high speed reaction time, expansion capability, priority and queueing versatility required for real-time control.

BUFFER INTERLACE CONTROLLER

Many system devices require computer facilities to transmit I/O data at high rates and volumes and at random periods. Such devices are best serviced with automatic channels which do not require programming or interfere with the processing. The Buffer Interlace Controller (BIC) unit option services such requirements.

The BIC contains two 15-bit registers, the Party Line addressing and control logic, priority logic, and 620/i control logic. The two

registers contain the stop address and the current memory address. These registers are set by the program with the start address and the stop address. These addresses define the sequential locations in memory from or to which the data is communicated. Connecting the desired controller to the BIC activates the BIC. The I/O operation is automatic thereafter until the stop address has been met. Each data word transferred requires less than two memory cycles. Information can be transferred at a rate over 200,000 words per second. The BIC automatically synchronizes the data transmission rate to the device requirement.

The BIC connects to the Party Line and controls the data transmission of the devices with BIC adapters when operating in the "interlace" mode. Interlace I/O occurs on a memory cycle basis and shares priority with the Control Processor. The BIC will capture the next memory cycle and stall the computer for 2.7 microseconds for each word transmitted. The processing resumes automatically at the completion of the word transferred. Any device connected to the BIC can be operated under control of the BIC or under program control. Up to eight devices can be connected to one Buffer Interlace Controller unit. The current address can be read under program control.

REAL-TIME CLOCK

The DATA 620/i Real-Time Clock is an option that provides a flexible time-orientation system that can be used in a variety of real-time functions, including time-of-day accumulation and as an interval timer.

The Real-Time Clock consists of two interrupts. The first interrupt is a time-base signal that when recognized by the computer, executes an increment memory instruction stored in the interrupt address. The second interrupt occurs

when the incremented memory location reaches a count of 40,000g.

SENSE LINE

Discrete sense lines are available as options in sets of eight. Each sense line has a unique address. Up to 512 sense lines can be addressed. The sense instruction is a two word conditional jump command. If a signal exists on the sense line addressed, the program jumps to the effective address; otherwise, the program continues at location $P + 2$. The sense lines can be configured in combination with the interrupt lines to permit more than one device to share an interrupt line. All DATA 620/i peripheral equipment include the sense lines required.

EXTERNAL CONTROL LINES

Discrete control lines are available as options in sets of eight. Each control line has a unique address. Up to 512 control lines can be addressed. The external control instruction is a one word instruction that places a pulse on the addressed control line. These are general purpose control lines that can be used to perform external control functions throughout a system. The control pulse has a 450 nanosecond width. The control lines required by DATA 620/i options are provided with the option.

PARALLEL I/O CHANNELS

The usual system application requires special devices to be connected to the computer. These devices can be interfaced with the computer in many ways. The system designer can implement the interface with his own electronics, purchase and assemble the appropriate logic modules (Micro-Versalogic), or utilize the Varian Data Machine interface controllers.

DIGITAL I/O

The interface controller provides the timing, gating and selection logic needed to communicate with the Party Line I/O lines under program control. The controller has:

- Eight Sense Lines
- Eight External Control Lines
- Buffered Input Channel — Provides an 18-bit register to receive pulsed inputs for subsequent input to the Party Line.
- Buffered Output Channel — Provides 18 stored logic levels (flip-flops) for level output from the Party Line.

The controller is 18-bit parallel (on the 16-bit computer, 2 bits are not used) and greatly alleviates the interface problem.

1.8 PERIPHERAL EQUIPMENT

A full line of compatible peripheral equipment is available for the DATA 620/i series. Each device has been selected to meet the functional requirements of a real-time data system.

Each piece of peripheral equipment is provided with a controller that includes a Party Line adapter, buffering and control lines. The line printer, disc storage, and magnetic tapes include word assembly/disassembly registers. The magnetic tape control units contain double buffers to permit multiple simultaneous high-performance magnetic tape operation.

The peripherals will operate with the Party Line under program control, or automatically with an (optional) Buffer Interlace Controller.

A complete line of analog conversion equipment is offered on a custom basis according to the requirement.

DATA 620/i SERIES PERIPHERAL EQUIPMENT

MAGNETIC TAPE SYSTEMS

Tape Controllers — Master controller for up to four tape transports. Will control 7 or 9 track transport and includes assembly/disassembly register.

Tape Transports — Speeds of 45, 75 and 120 ips. Densities of 200, 556 and 800 bpi. Seven and nine track industry compatible units.

AUXILIARY STORAGE

Fixed head rotating memory systems with capacities from 32K words to 262K words. Access time is 17 milliseconds. Transfer rates from 30 to 120 kbps.

READERS AND PUNCHES

Card Reader — 1000 cpm
Paper Tape Reader — 300 cps
Paper Tape Punch — 60 and 120 cps

DIGITAL INPUT/OUTPUT

KEYBOARD

ASR 33 Teletypewriter
ASR 35 Teletypewriter
KSR 35 Teletypewriter

GRAPHIC DEVICES

Oscilloscope Displays
High Speed Printers — 300 and 600 lpm

Electrostatic Plotters
Digital Plotters — 300 steps per sec.

MODEM INTERFACES

103, 201, and 301 types

Detailed design and maintenance information on peripheral device controllers is contained in individual reference manuals for these units. Operation and maintenance procedures for optional peripheral devices (tape transports, printers, etc.) are contained in the manufacturers' reference manuals furnished with the equipment.

1.9 SYSTEM SOFTWARE

A comprehensive package of operational programs are available with the DATA 620/i. These include a symbolic assembler, FORTRAN compiler, library of mathematical subroutines, debugging package, and a modular maintenance diagnostic package. The complete software package operates in the basic 8,192 words of core memory. In addition, Varian Data Machines has developed many real-time programs for a specific customer application. The more important portions of the Varian Data Machine software library are described briefly below, and in detail in sections 10 through 12.

SYMBOLIC ASSEMBLER

The DATA 620/i Assembler System (DAS) is a two-pass assembler that assists in program preparation by allowing instructions, addresses, etc., to be specified in a straight-forward and meaningful manner. DAS recognizes over 20 pseudo-operations that aid the user in coding and debugging problems. Although DAS operates in a minimum system consisting of 4,096

words of core memory, paper tape reader, paper tape punch and typewriter, provisions have been made to utilize additional memory and peripheral equipment available to the system. Extensive syntax checking is performed during both passes of the assembler. For a complete description of DAS, see section 10.

FORTRAN

DATA 620/i FORTRAN conforms with the proposed American standards for Basic FORTRAN as published by the American Standards Association. The 620/i FORTRAN, a one-pass compiler, can operate in a 8,192 word computer equipped with only a model ASR 33 Teletypewriter. Naturally, if higher performance peripherals are on the system, 620/i FORTRAN utilizes them to produce faster compilation. For a complete description of FORTRAN, see section 12.

AID

AID is a collection of useful diagnostic and utility routines for the DATA 620/i computer. With this package, the programmer can call upon a wide variety of functions to aid him in debugging and running his programs. AID includes routines to correct memory, establish breakpoints, search memory, print memory, etc.

Also included in the AID package is a comprehensive binary paper tape handler that is particularly useful in preserving programs modified on the computer. This routine uses a standard address, data, and checksum format that is used by the DAS assembler. AID is described in section 11.6.

DIAGNOSTIC PROGRAM PACKAGE

The DATA 620/i Diagnostic Program Package is designed to check instructions, memory and input/output devices, and to isolate errors. It can be used in either the preventive or the corrective mode of operation. In the preventive mode, the complete system is checked for operational readiness. If a malfunction exists, in most cases, the preventative will isolate the error. The corrective mode of operation is used when a malfunction is known to exist and the preventive mode does not decisively show the trouble. Proper application of these diagnostic routines can cut the mean-time-to-repair to minutes. This modular package can be easily expanded to accommodate any special system hardware tests.

SUBROUTINE LIBRARY

This comprehensive library includes the most commonly used subroutines needed in a systems environment. The library includes routines for logarithmic, exponential and trigonometric functions, for fixed and floating-point arithmetic, and for operating standard peripheral equipment. Conventions and instructions are provided so the user can add application programs to the library and be called by DAS, FORTRAN and AID. The individual subroutines are described in section 11.

1.10 USER SERVICES

The purchase of a DATA 620/i includes support services designed to provide the user with start-up and sustaining service.

DOCUMENTATION

The documentation is comprehensive and clear, and contains the information required for the user to fully understand, program, operate

and maintain the system. Interface and installation manuals are provided to the user prior to installation for system integration preparation. The program and service manuals are provided in advance of the user training attendance. The software manuals contain a special section covering software modularity and expansion techniques.

PROGRAMMING TRAINING*

Programming training courses are provided on a scheduled basis at Varian Data Machine facilities. The one week course covers instruction for programming in machine language, an introduction to the DATA 620/i software, and machine operation. The course includes time at the console. Supplies required for the course are provided at no charge to the attendees. On-site courses are available on a contract basis.

MAINTENANCE TRAINING*

A two-week-at-the-factory maintenance course is provided on a scheduled basis. The instruction covers machine organization, operation, logic, design, timing, preventive maintenance, trouble-shooting and repair. Extended training covering special systems hardware is available on an individual customer basis. The course is designed for personnel with existing digital logic design knowledge.

USER ORGANIZATION

Varian Data Machine Customer Services (CS) provide continuing coordination, program exchange and library maintenance for DATA 620 and DATA 620/i users. Users are notified of

* Available at nominal cost.

new additions to the library, application data, program and hardware modifications and new equipment. CS maintains up-to-date master prints on each system controlled. An inventory of programming forms, paper tapes and spare parts is maintained for expedited or emergency service. Statistical data on field operating experience based on user-submitted reports is maintained and available to users. On-call and on-site maintenance services are available on a contract basis.

APPLICATION PROGRAMMING

Varian Data Machines' technical staff includes senior application programming specialists well-qualified to assist the user in the preparation of application programs. This professional group can assume full responsibility on a contract basis for the preparation of a total solution, including hardware and application programs.

1.11 DATA 620/i SPECIFICATIONS

TYPE

A system computer, general purpose digital, designed for on-line data system requirements, magnetic core memory, binary, parallel, single-address, with bus organization and micro-control.

MEMORY

Magnetic Core, 16 bits (18 bits optional), 1.8 microseconds full cycle, 750 nanoseconds access time, 4096 words minimum expandable to 32,768 words.

ARITHMETIC

Parallel, binary, fixed point, 2's complement.

WORD LENGTH

16 bits standard; 18 bits optional.

SPEED (FETCH & EXECUTE)

Add or Subtract:

3.6 microseconds.

Multiply (optional):

18.0 microseconds, 16-bit.

19.8 microseconds, 18-bit.

Divide (optional):

18.0 to 25 microseconds, 16-bit.

19.8 to 28.8 microseconds, 18-bit.

Register change class:

1.8 microseconds.

Input/Output — from A or B register:

3.6 microseconds.

Input/Output — from memory:

5.4 microseconds.

OPERATION REGISTERS

A Register — accumulator, input/output, 16/18 bits.

B Register — double length accumulator, input/output, index register, 16/18 bits.

X Register — index register, 16/18 bits.

P Register — program counter, 16/18 bits.

BUFFER REGISTERS

R Register — operand register, 16/18 bits.

U Register — instruction register, 16 bits.

S Register — shift register, 5 bits, operates with the U Register for executing shift instructions.

L Register — memory address register, 16 bits.

W Register — memory word register, 16/18 bits.

CONTROL

Addressing Modes

Direct addressing to 2,048 words

Relative to P Register 512 words

Index with X Register, hardware, does not add to execution time

Index with B Register, hardware, does not add to execution time

Multi-level indirect addressing

Immediate

Extended addressing (optional)

Instruction Types

Single word

Double word

Instructions: Over 100 standard commands, listed below, plus more than 128 macro-instructions:

- 3 Load
- 3 Store
- 5 Arithmetic (2 optional)
- 3 Logical
- 10 Jump
- 10 Jump and Mark
- 10 Execute
- 14 Immediate (2 optional)

13 Input/Output

26 Register Change

6 Logical Shift

6 Arithmetic Shift

2 Control

14 Extended addressing (optional)
Over 128 micro-instructions

MICRO-EXEC (Optional)

Facility and hardware to construct a hardware program external to the DATA 620/i. Eliminates stored program memory accessing by use of hardware program.

CONSOLE

Display and data entry switches for all operational registers, 3 SENSE switches, instruction REPEAT, single STEP, RUN, POWER on/off.

INPUT/OUTPUT

Processor Input/Output Options

Programmed Data Transfer

Single word to/from memory

Single word to/from A and B registers

External control lines

External sense lines

Automatic Data Transfer

Direct memory access facility transfer with rates over 200,000 words per second.

Priority Interrupts

Group enable/disable, individually arm/disarm, single instruction interrupt capability.

Real-Time Clock

Adjustable time base: May be programmed as multiple internal timers.

Power Failure Detect/Restart

Interrupts on power failure and automatically restarts on power recovery.

PHYSICAL

Dimensions

Mainframe—10-1/2 inches high, 19 inches wide, 15 inches deep.

Weight

Mainframe — 35 pounds.

Power

3 amps (340 watts). 115 $\pm$ 10v, 60 $\pm$ 2Hz. Power supplies are regulated. Additional regulation is not required under normal commercial power sources.

Conversion for 50Hz and other voltages available at added cost.

EXPANSION

Main processor contains provisions and space for all internal options.

INSTALLATION

Mounts in standard 19-inch cabinet, no air conditioning, sub-flooring or special wiring and site preparation required.

ENVIRONMENTS

5°C to 45°C; 0 to 90% relative humidity.

MAINFRAME LOGIC AND SIGNALS

Integrated circuit, 8.8 MHz clock, logic levels 0v false, +5v true.

FULLY COMPATIBLE SYSTEM COMPONENTS

To increase your total system capability, Varian Data Machines offers a complete line of high-performance integrated circuit logic

modules, small high-speed core memories and large mainframe memories for I/O equipment or additional system requirements. All have been field-proven with the DATA 620/i System, and are fully compatible with its power supply, voltage levels and signal requirements.

MICROVersaLOGIC INTEGRATED CIRCUIT LOGIC MODULES

MicroVersaLOGIC 5 m.c. general purpose IC modules — with NAND, NOR logic, and wired OR capability at the collector—5v logic levels —and excellent noise rejection over 1v. Over 25 module types, including universal flip-flops, delay multivibrators, clock drivers, 2-, 3- and 4-input expandable gates and PNP to NPN interface modules. Compatible mounting hardware — including card files and card drawers — is also available.

VersaSTORE II CORE MEMORIES

New High-speed core memory systems — with integrated circuits and all-silicon components for highest reliability — that operate asynchronously at 1.6 microseconds, with 650 nanosecond access time. VersaSTORE II memories are available in increments up to 4,096 words of 36 bits, require only 5-1/4 inch of rack space, and can also be provided as 8k word memories of up to 18 bits.

Options include parity line, built-in self-test, and a variety of timing and control flags.

VersaSTORE MAINFRAME MEMORIES

High-reliability VersaSTORE mainframe memories in sizes up to 65k words in 4k increments, with word lengths to 36 or 72 bits. Features include PNP to NPN interface — flexible input levels of 3v to 12v — continuous lamp display of address and data registers — servoed current drive — 2 sec operation — integrated circuit design — and DATA GUARD protection system.

SECTION 2

SYSTEM DESCRIPTION

2.1 COMPUTER ORGANIZATION

The DATA 620/i is organized with a unique bus structure, selection logic and nine registers. The organization provides universal information routing, buffered processing, microprogramming capability, indexing without time penalty and buffered input/output data transfer. A unique optional facility, Micro-EXEC, is also available which permits complex algorithms to be implemented with external control hardware. This capability provides increases in processing speed in excess of 400 percent over normal programmed operations.

The organization of the DATA 620/i is shown in Appendix A. This diagram shows the major functional elements of the machine, including the registers and buses provided for information transfer.

The major functional elements of the DATA 602/i, indicated in figure A-1, are: control section, arithmetic/logic section, operational registers, internal buses, input/output (I/O) bus and memory.

Control Section

The control section provides the timing and control signals required to perform all operations in the computer. The major elements in this section are the U register, the timing and decoding logic and the shift control.

The U register (instruction register) is 16 bits long. This register receives each instruction

from memory through the W bus and holds the instruction during its execution. The control fields of the instruction word are routed to the decoding and timing logic where the codes determine the required timing and control signals. The address field from the U register, used for various addressing operations, is also routed to the arithmetic/logic section.

The decoding logic decodes the fields of the instruction word held in the U register to determine the control signal levels required to perform the operations specified by the instruction. These levels select the timing signals generated by the timing unit.

Timing logic generates the basic 4.4-MHz system clock. From this clock, timing logic derives the timing pulses which control the sequence of all operations in the computer.

The shift control contains the shift counter and logic to control operations performed by the shift, multiply and divide instructions.

Arithmetic/Logic Section

This section consists of two elements; the R register and the arithmetic unit.

The R register receives operands from memory and holds them during instruction execution. The operand may be either data or address words. This register permits transfers between memory and I/O bus during the execution of extended-cycle instructions.

The arithmetic unit contains gating required for all arithmetic, logic and shifting operations performed by the computer. Indexed and relative address modifications are performed in this section without increased instruction execution time.

The arithmetic unit also controls the gating of words from the operational registers and the I/O bus onto the C bus where they are distributed to the operational registers or to memory registers. This facility is used to implement many of the microinstructions of the computer.

Operational Registers

The basic DATA 620/i computer contains nine registers.

The operational registers consist of the A, B, X and P registers. The A, B and X registers are directly accessible to the Programmer. The P register is indirectly accessible through use of the jump-class instructions which modify the program sequence. The operational registers are described in the following paragraphs.

A register. This full-length register is the upper half of the accumulator. This register accumulates the results of logical and addition/subtraction operations, the most-significant half of the double-length product in multiplication, and the quotient in division. It may also be used for input/output transfers under program control.

B register. This full-length register is the lower half of the accumulator. This register accumulates the least-significant half of the double-length product in multiplication, and the quotient in division. It may also be used for input/output transfers under program control and as a second hardware index register.

X register. This full-length register permits indexing of operand addresses without adding time to execution of indexed instructions.

P register. This full-length register holds the address of the current instruction and is incremented before each new instruction is fetched. A full complement of instructions is available for conditional and unconditional modification of this register.

Auxiliary Registers

The other registers and their uses are described below:

U register is a full-word buffer which holds the instruction being executed. The U register buffers the control unit from memory to permit interlace I/O operation to occur on a memory cycle by memory-cycle-basis. It is also a multipurpose register available to Micro-EXEC.

S register is a 5-bit register which, in combination with the U register controls the length of shift instructions. This register also buffers memory from the control unit. S register is available to Micro-EXEC.

L register is the 16-bit memory location register. Micro-EXEC can select and set the L register.

W register is the memory word register and is full length (16 or 18 bits). W is selectable and can be set by Micro-EXEC.

R register is a full-word buffer which holds the multiplicand and divisor, in arithmetic operations. R register buffers the arithmetic unit from memory to permit interlace I/O operations to occur on a memory-cycle-steal basis. It is also a multipurpose register available to Micro-EXEC.

Internal Buses

The basic computer contains five buses. These are the C, S, W, L and I/O buses, described in the following paragraphs.

C bus. This bus provides the parallel path and selection logic for routing data between the arithmetic unit, the I/O bus, the operational registers and the memory registers. The console display indicators are also driven from the C bus. Distribution of data simultaneously to multiple operational registers is facilitated by this bus.

S bus. This bus provides the parallel path and selection logic for routing data from the operational registers to the arithmetic unit.

W bus. The memory word (W) register is directly connected to all memory modules through the W bus. The bus is bidirectional and time-shared among memory modules.

L bus. The memory address (L) register is directly connected to all memory modules through the L bus. The bus is unidirectional.

Input/Output Bus

The standard DATA 620/i is provided with a bidirectional input/output (I/O) bus that permits programmed data transfers between peripheral devices and the computer. This bus is described in more detail in section 6.2.

Memory

The internal storage of the computer consists of 4096-word modules connected to the L and W buses. The mainframe can accommodate one 4906-word module. Additional

modules are added in an additional frame that is attached to the mainframe. The computer memory can be expanded to a maximum of 32,768 words using 4096-word modules. Instruction words read from memory are transferred to the control section for execution. Words may be transferred, under program control, from memory to the arithmetic/logic section, to the operation registers, or to the I/O bus. Words may be transferred, under program control, to memory from the operational registers or the I/O bus. When the direct memory access option is used, the system is capable of direct transfer between memory and peripheral devices on the I/O bus, concurrent with computations.

Direct Memory Access

The direct-memory-access (DMA) option (section 7) allows data transfer into or out of memory modules without disturbing the contents of the operational registers. Only the L and W registers are altered. Access to memory using the DMA facility is on a "cycle-steal" basis and required 2.7 microseconds of processor time per transfer.

Micro-EXEC

The Micro-EXEC option (section 9.10) is a unique hardware technique for microstep sequencing of the computer. This option provides hardware logic in which all computer control signals are made available on an external cable connector so that special hardware routines can be constructed. External control and special return instructions are provided for easy program entry and exit.

2.2 COMPUTER WORD FORMATS

The Micro-EXEC has access to all registers and can manipulate the data flow between them in the same manner as the basic computer control. There are three basic word

formats used in the DATA 620/i: data, indirect address, and instruction. The instruction word format is further divided into four types: single-word addressing, single-word non-addressing, double-word addressing, and double-word non-addressing.

Data Word Format

The data word format is shown in figure 2-1. This word may be either 16 or 18 bits depending upon the word length configuration of the particular machine.

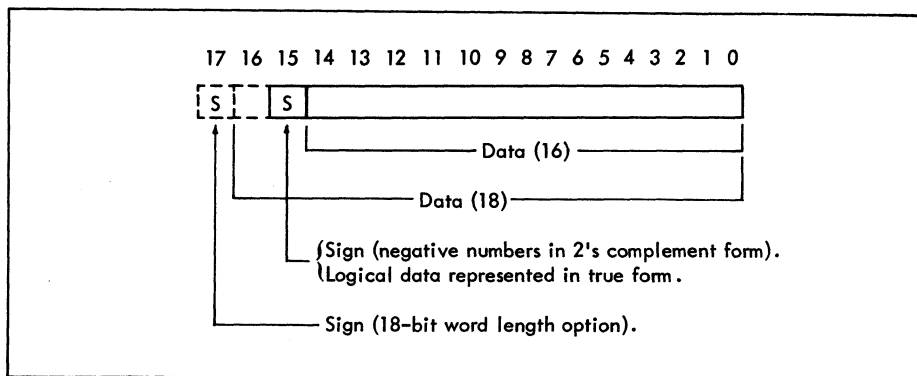


Figure 2-1. Data Word Format

In the 16-bit format, the data occupy bit positions 0-14, with the sign in position 15. Negative numbers are represented in 2's complement form. In the 18-bit format, the data occupy bits 0-16, with the sign in position 17.

Address Word Format

The address word format is shown in figure 2-2. This word occupies a location in memory which is accessed by an instruction in the

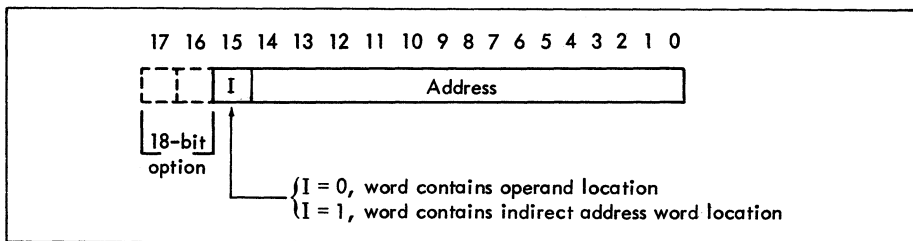


Figure 2-2. Address Word Format

indirect address mode or it is the second word of a double-word instruction. Bit 15 contains the I Bit. If I = 0, bits 0-14 contain the location of an operand or instruction in memory. If I = 1, bits 0-14 contain the location of another indirect address word. Indirect addressing may be extended to any desired level. Each level of indirect addressing adds one cycle (1.8 microseconds) to the basic execution time of an instruction.

Single-Word Instruction Formats

Single-word instructions may be either addressing or non-addressing, as described below and in section 3.1.

Addressing instructions. The single-word addressing instruction format is shown in figure 2-3. This type of word contains three fields, as follows:

O - Operation code

m - Addressing mode

M - Memory Address field

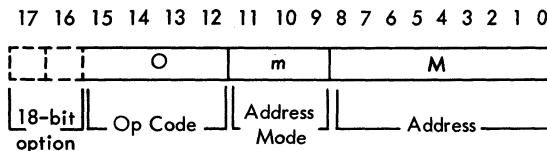
All single-word addressing instructions may be executed in any one of five addressing modes: direct, relative to P register, index with X register, index with B register and indirect.

Single-word addressing instruction groups are as follows:

LOAD/STORE

ARITHMETIC

LOGICAL



M Field:	0XX - Direct	operand in location 0 - 2047 (bits 10 to 0).
	100 - Relative	add A field to P register.
	101 - Index (X)	add A field to X register.
	110 - Index (B)	add A field to B register.
	111 - Indirect	stored at A field location.

Figure 2-3. Single-Word Addressing Instruction Format

Non-addressing instructions. The single-word non-addressing instruction format is shown in figure 2-4. This instruction contains the following three fields:

- C - Class code
- O - Operation code
- D - Definition

The D (definition field) specifies the action to be performed by the computer such as:

- a. Number of shifts.
- b. Kind of register change as well as source and destination registers.
- c. Input/output.
- d. Halt code.

Single-word non-addressing instruction groups are as follows:

SHIFT
CONTROL
REGISTER CHANGE
INPUT/OUTPUT

Double-Word Instruction Formats

Double-word instructions may be either addressing or non-addressing as described below and in Section 3.2.

Addressing Instructions. This instruction contains three fields:

- C - Class code
- O - Operation code
- D - Definition

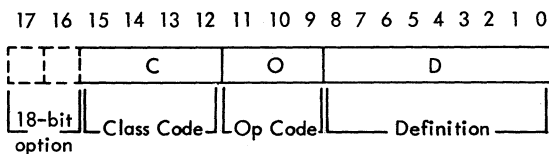


Figure 2-4. Single-Word Non-Addressing Instruction Format

The double-word addressing instruction is shown in figure 2-5. This format is used for the following instruction types:

JUMP
JUMP AND MARK
EXECUTE
EXTENDED ADDRESS

For the jump, jump-and-mark, and execute groups, the definition field of the first word defines a set of nine logical states which condition the execution of the instruction. The second word contains the jump address, jump-and-mark address, or the location of the instruction to be executed if the condition is met. Indirect addressing is permitted.

For the extended address group of instructions, the definition field is further divided into three subfields. The m field contains bits 0-2, the op code contains bits 3-6, with bits 7 and 8 left blank. Extended address instructions are identical in operation to the single-word addressing instructions having the same op code except that they allow direct addressing to 32,768 words of memory.

For the memory input/output group, the definition field of the first word contains the number of the peripheral device and its mode, and the second word contains the memory address of the data to be transferred. Indirect addressing is not permitted.

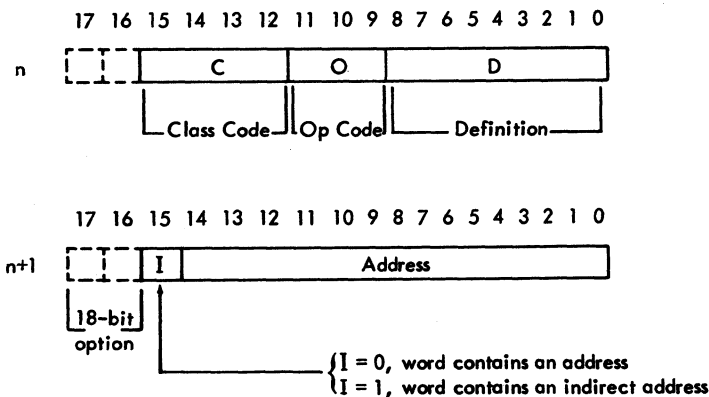


Figure 2-5. Double-Word Addressing Instruction Format

Non-addressing instructions. The double-word non-addressing instruction format is shown in figure 2-6. This format is used for the immediate group of instructions. There are 12 standard and two optional instructions in this group.

The op-code field contains the operation to be performed (bits 3-6). All single-word addressing type instructions may be performed as an immediate type instruction. The operand is contained in the second word. Indirect addressing is not applicable.

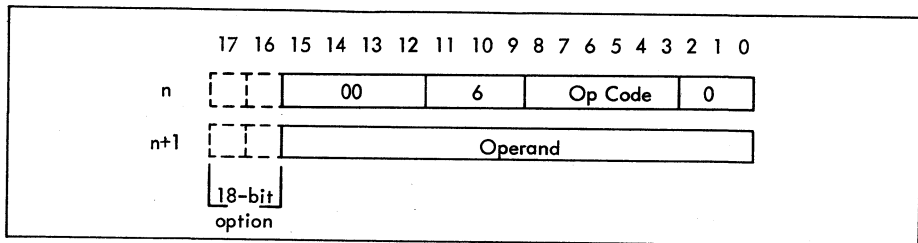


Figure 2-6. Double-Word Instruction Format, Immediate Type Instructions

SECTION 3

OPERATIONAL INSTRUCTIONS

This section describes DATA 620/i instructions which effect operations in the computer. Input/output instructions are described in section 4. Information provided for each instruction is as follows:

- a. Mnemonic code that is recognized by the DATA 620/i assembler (DAS).
- b. The octal digit configuration of the instruction.
- c. Instruction timing.
- d. Instruction description.
- e. Registers altered by execution of the instruction.
- f. Addressing modes permitted.
- g. A flow chart, when required for complete understanding.

Instructions are divided into two classes: single-word and double-word. Each class contains both addressing and non-addressing groups of instructions. Microprogramming operations which can be implemented for various instruction types are summarized in Appendix C tables C-3 through C-10.

3.1 SINGLE-WORD INSTRUCTIONS

Single-word instructions may be either addressing or non-addressing. The addressing instruction groups are load/store, arithmetic (multiply/divide optional) and logical.

The non-addressing instruction groups are control, shift and register change.

Single-Word Addressing Instructions

The format of the single-word addressing class instructions is shown in figure 2-3. The operation is specified by the O field (bits 12-15). The memory address field, M (bits 0-8), contains the base location of an operand in memory. Operand addressing may be in any one of five modes specified by the m field (bits 9-11).

Table C-3 summarizes the operation codes for the single-word addressing instructions. Table C-3(d) illustrates the addressing modes that can be used. Figure 3-1 shows the general operand addressing flow for this class of instructions.

For direct addressing, bits 0-10 specify the location of an operand within the first 2048 (0-2047) words of memory.

For relative addressing, the address field, mod 2^9 , is added to the P register to form the effective address. This mode permits addressing an operand up to 512 words in advance of the current program location.

For index addressing with the X or B register, the address field is added to the X or B register, mod 2^{15} , to form the effective address. Indexing does not increase the basic instruction execution time.

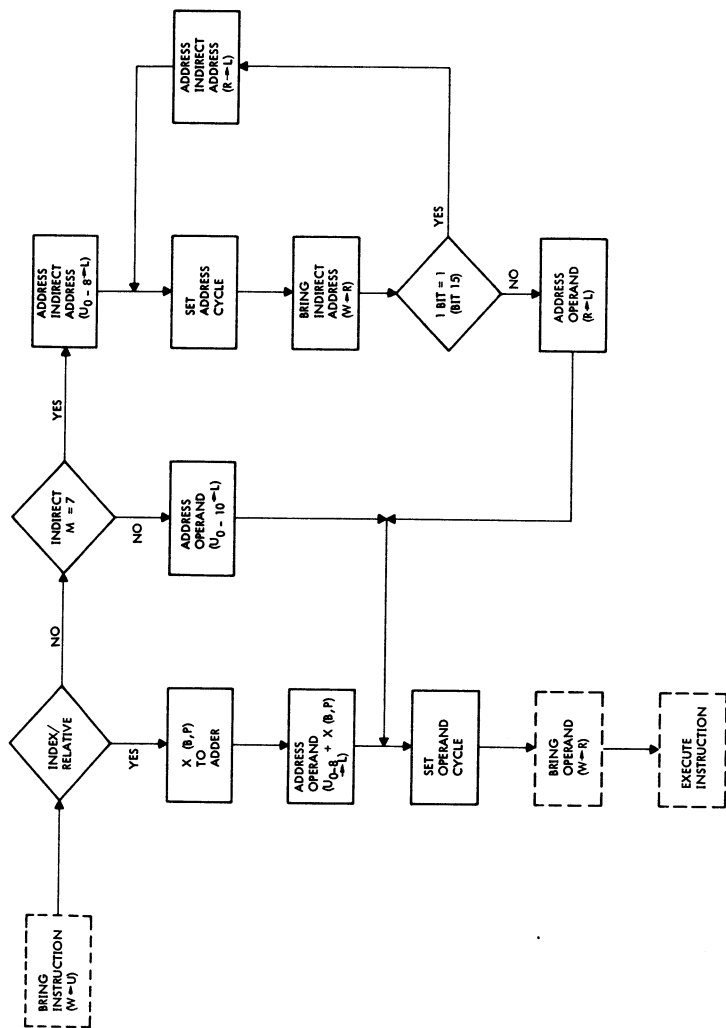
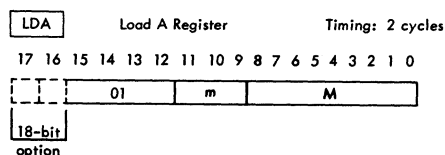


Figure 3-1. Single-Word-Address Instruction, Operand Addressing, General Flow

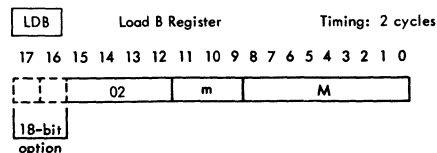
For indirect addressing, the address field specifies the location of an indirect address word within the first 512 (0-511) words of memory. If $I = 0$ in the address word, the word contains the location of an operand. If $I = 1$, the word specifies the location of another indirect address word. Each level of indirect addressing adds one cycle (1.8 microseconds) to the basic instruction execution time.

Load/store instruction group (table C-3(a)). The following paragraphs provide the mnemonic, description and timing for each instruction in the load/store group. Figures 3-2 and 3-3 show the general flow for the load/store instruction group.



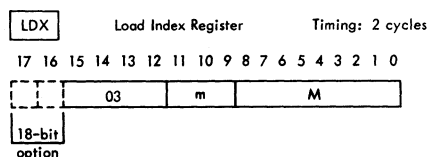
The contents of the addressed memory location are placed in the A register.

Relative: Yes
Indexing: Yes
Indirect Addressing: Yes
Registers Altered: A



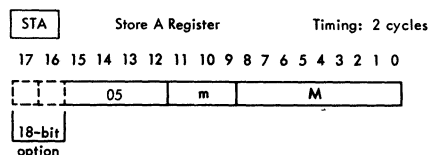
The contents of the effective memory location are placed in the B register.

Relative: Yes
Indexing: Yes
Indirect Addressing: Yes
Registers Altered: B



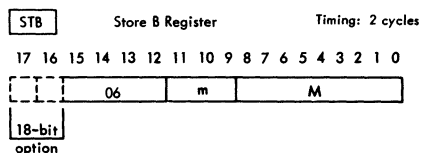
The contents of the effective memory location are placed in the index register.

Relative: Yes
Indexing: Yes
Indirect Addressing: Yes
Registers Altered: X



The contents of the A register are placed in the effective memory location.

Relative: Yes
Indexing: Yes
Indirect Addressing: Yes
Registers Altered: Memory



The contents of the B register are placed in the effective memory location.

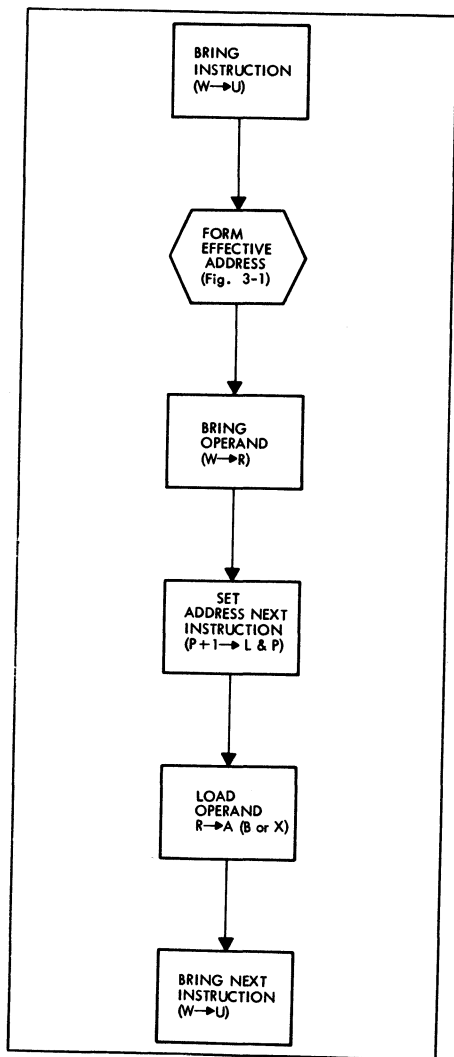


Figure 3-2. Load Type Instruction, General Flow

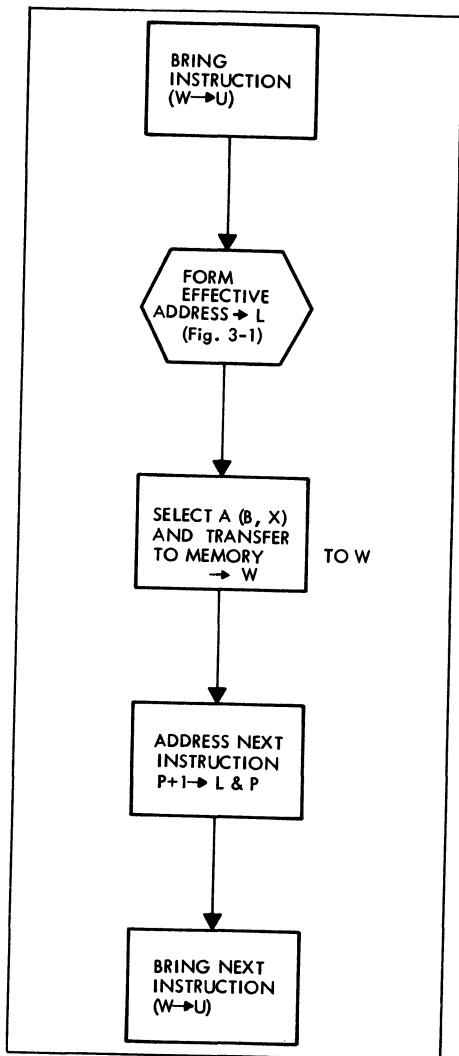
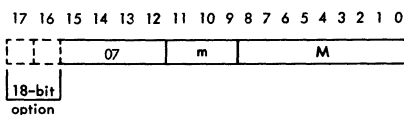


Figure 3-3. Store-Type Instruction, General Flow

Relative: Yes
 Indexing: Yes
 Indirect Addressing: Yes
 Registers Altered: Memory

STX Store Index Register Timing: 2 cycles

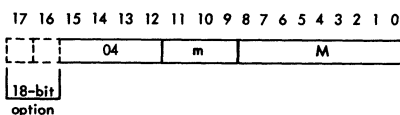


The contents of the X register are placed in the effective memory location.

Relative: Yes
 Indexing: Yes
 Indirect Addressing: Yes
 Registers Altered: Memory

Arithmetic instruction group. The following paragraphs provide the mnemonic, description, and timing for each instruction in the arithmetic group. Figures 3-4 and 3-5 show the general flow for the arithmetic instruction group.

INR Increment Memory and Replace Timing: 3 cycles

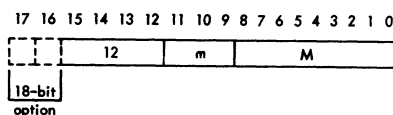


The contents of the effective memory location are incremented by one, mod 2^{16} (2^{18}).

After execution, if $(M) \geq 2^{15}$ (2^{17}), the overflow indicator (OF) is set.

Indexing: Yes
 Indirect Addressing: Yes
 Registers Altered: Memory, OF

ADD Add Memory to A Timing: 2 cycles

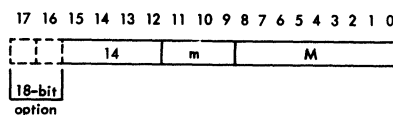


The contents of the effective memory location are added to the contents of the A register and the sum is placed in the A register.

After execution, if $(A) \geq 2^{15}$ (2^{17}) or $(A) < -2^{15}$ (-2^{17}), the overflow indicator (OF) is set.

Indexing: Yes
 Indirect Addressing: Yes
 Registers Altered: A, OF

SUB Subtract Memory from A Timing: 2 cycles

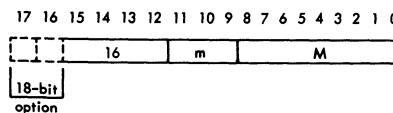


The contents of the effective memory location are subtracted from the A register and the difference is placed in the A register.

After execution, if $(A) \geq 2^{15}$ (2^{17}) or $(A) < -2^{15}$ (-2^{17}), the overflow indicator (OF) is set.

Indexing: Yes
 Indirect Addressing: Yes
 Registers Altered: A, OF

MUL Multiply (optional) Timing: 10 cycles (16 bits)
 11 cycles (18 bits)



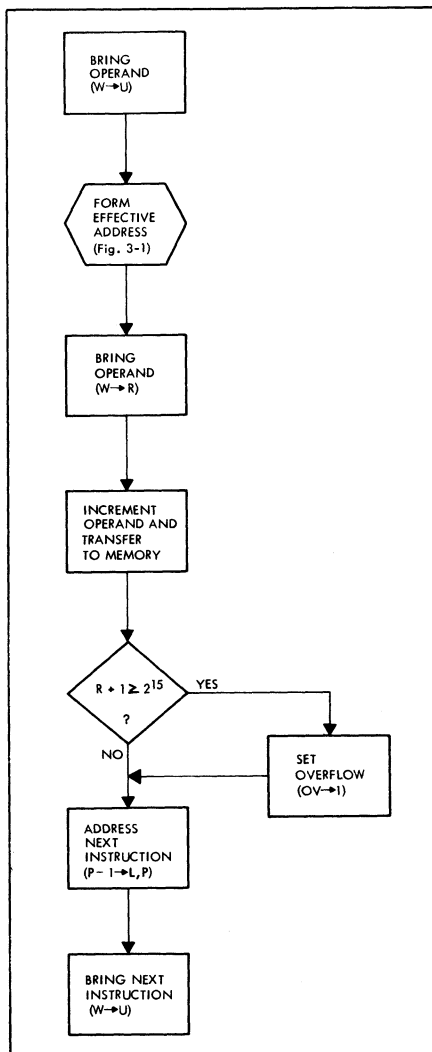


Figure 3-4. Increment Memory-and-Replace Instruction, General Flow

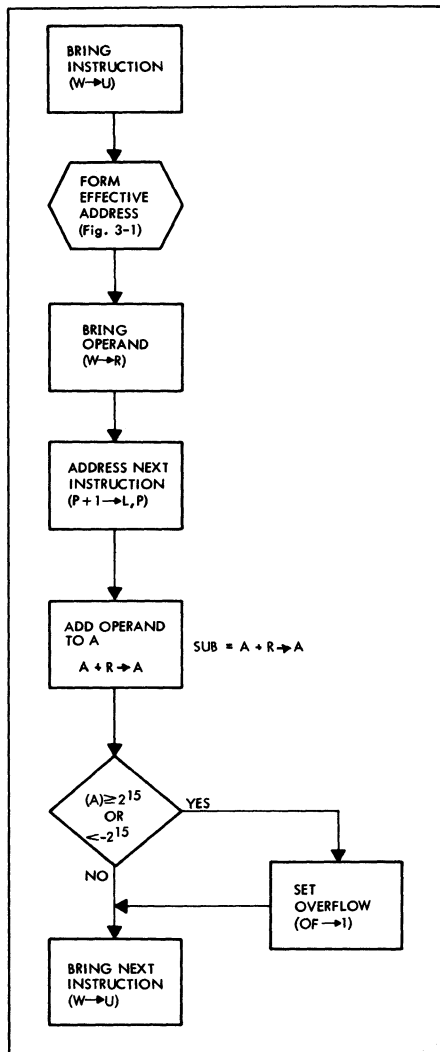


Figure 3-5. Add Instruction, General Flow

The contents of the B register are multiplied by the contents of the effective memory location. The original contents of the A register are added to the final product. The product is placed in the A and B registers, with the most-significant half of the product in the A register and the least-significant half in the B register. The sign of the product is contained in the sign position of the A register. The sign position of the B register is reset to zero.

The algorithm is in the form:

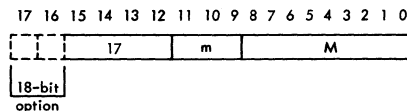
$$(M) \cdot (B) + (A) \rightarrow (A, B)$$

Indexing: Yes

Indirect Addressing: Yes

Registers Altered: A, B, OF

DIV	Divide (optional)	Timing: 10-14 cycles (16 bits) 11-15 cycles (18 bits)
------------	-------------------	--



The contents of the A and B registers are divided by the contents of the effective memory location. The quotient is placed in the B register with sign, and the remainder is placed in the A register with the sign of the dividend.

The algorithm is of the form:

$$(A, B) / (M) \rightarrow \begin{cases} (B) \text{ quotient} \\ (A) \text{ remainder} \end{cases}$$

If $|(A, B) / (M)| < 1$

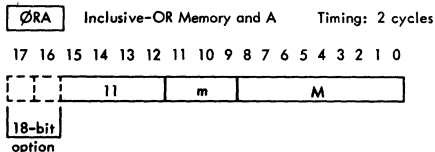
(|divisor| > |dividend|), taken as a binary fraction, overflow will not occur. If overflow does occur, the overflow indicator (OF) is set.

Logical instruction group. The following paragraphs provide the mnemonics, description, and timing for each instruction in the logical instruction group.

Indexing: Yes

Indirect Addressing: Yes

Registers Altered: A, B, OF



An inclusive-OR operation is performed between the effective memory location and the contents of the A register. The result is placed in the A register. If either the effective memory location or A contains a one in the same bit position, a one is placed in the result. The truth table is shown below, where n = bit position.

Condition		Result
$A_{(n)}$	Effective Memory Location (n)	$A_{(n)}$
0	0	0
0	1	1
1	0	1
1	1	1

Indexing: Yes

Indirect Addressing: Yes

Registers Altered: A

ERA Exclusive-OR Memory and A Timing: 2 cycles

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

13 m M

18-bit
option

An exclusive-OR operation is performed between the effective memory location and the contents of the A register. The result is placed in the A register. If the same bit position of the effective memory location and A contain a zero, or if both bit positions contain a one, the result is zero. If the same bit position of the effective memory location and A are not equal; i.e., one contains a zero and the other a one the result is a one. The truth table is shown below, where $n = \text{bit position}$:

Condition		Result
$A_{(n)}$	Effective Memory Location (n)	$A_{(n)}$
0	0	0
0	1	1
1	0	1
1	1	0

Indexing: Yes
Indirect Addressing: Yes
Registers Altered: A

ANA AND Memory and A Timing: 2 cycles

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

15 m M

18-bit
option

The logical-AND is performed between the contents of the A register and the contents of the effective memory location. The result is placed in the A register. If the same bit position of both the effective memory location and A contain a one, the result is a one. The truth table is shown below, where $n = \text{bit position}$:

Condition		Result
$A_{(n)}$	Effective Memory Location (n)	$A_{(n)}$
0	0	0
0	1	0
1	0	0
1	1	1

Indexing: Yes
Indirect Addressing: Yes
Registers Altered: A

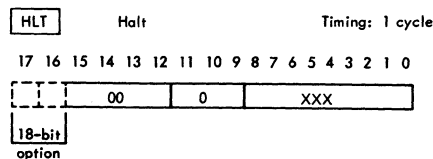
Single-Word Non-Addressing Instructions

The format of the single word non-addressing instruction class is shown in figure 2-4.

The non-addressing single-word instructions include the control group, the shift group, and the register change group. The operation is defined by the m field. The address field (M) is not used by the control group instructions. For the shift group, the M field defines the type and number of shifts. For the register change group, the M field defines the type of transfer and the registers affected.

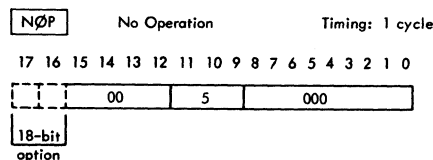
Control instruction group. The following paragraphs provide mnemonic, description and timing for each instruction in the control

group. Table C-4, Appendix C, summarizes the control instructions.



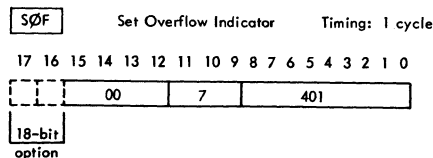
When the computer executes the halt instruction, computation is stopped and the computer is placed in the step mode. When the RUN button is pressed, computation starts with the next instruction in sequence.

Indexing: No
Indirect Addressing: No
Registers Altered: None



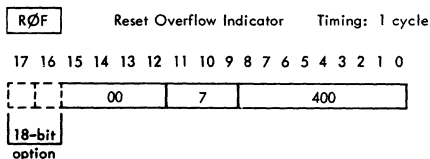
Execution of the NOP instruction does not affect the A, B, X registers or memory.

Indexing: No
Indirect Addressing: No
Registers Altered: None



The overflow indicator (OF) is set.

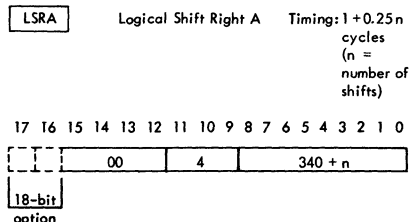
Indexing: No
Indirect Addressing: No
Registers Altered: OF



The overflow indicator (OF) is reset

Indexing: No
Indirect Addressing: No
Registers Altered: OF

Shift instruction group. For shift instructions the memory address field, M, defines the type of shift (bits 5-8) and the number of bit positions to be shifted (bits 0-4). The instruction format showing the use of each M-field bit is given in table C-3 of Appendix C. Twelve of the possible sixteen shift operations defined by bits 5-8 are implemented. These are summarized in table C-3(b). Figure 3-6 shows the general flow for the shift instructions.



The contents of the A register are shifted n places to the right (n = 0 to 378). Zeros are shifted into the high-order positions of the A register. Information shifted out of the low-order position of the A register is lost.

Indexing: No
Indirect Addressing: No
Registers Altered: A

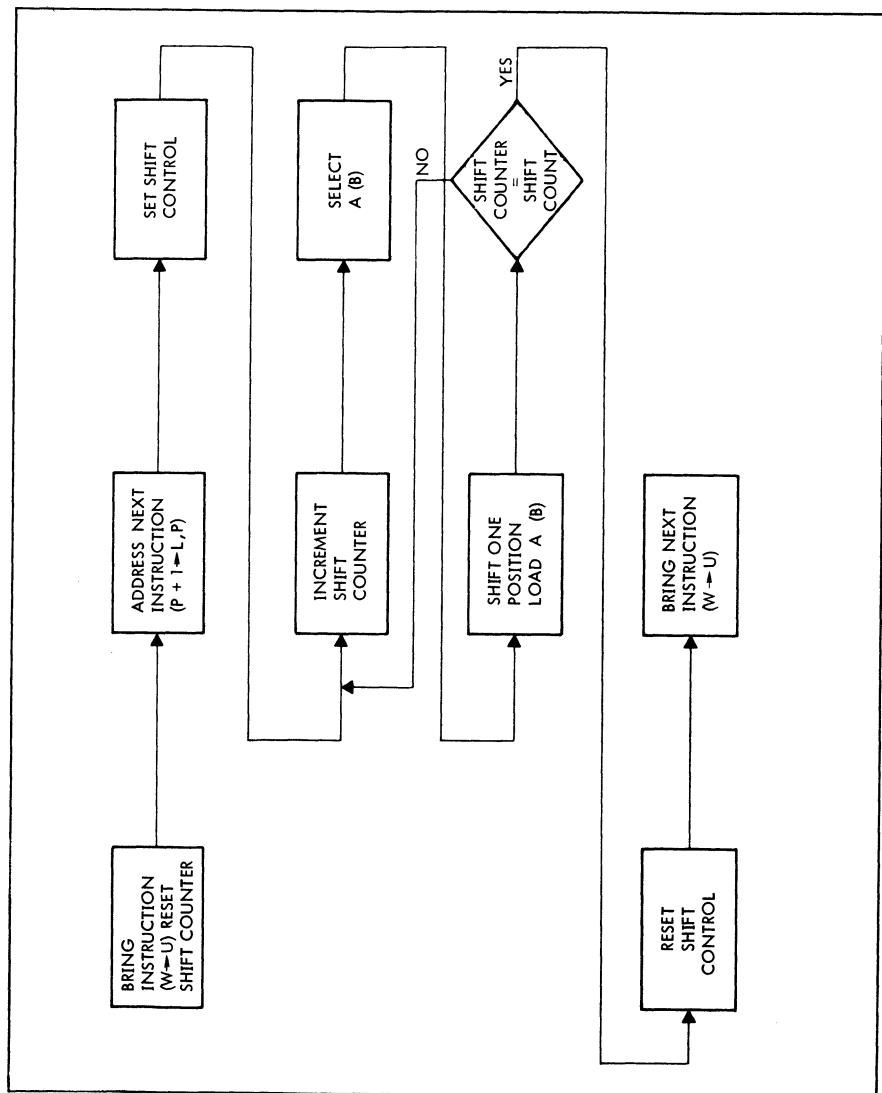


Figure 3-6. Single-Register Shift Instruction, General Flow

LSRB Logical Rotate Right B Timing: $1 + 0.25 n$ cycles
(n = number of shifts)

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

00	4	$140 + n$
----	---	-----------

18-bit
option

The contents of the B register are shifted n places to the right ($n = 0$ to 378). Information shifted out of the low-order position of the B register is lost. Zeros are shifted into the high-order position of the B register.

Indexing: No
Indirect Addressing: No
Registers Altered: B

LRLA Logical Rotate Left A Timing: $1 + 0.25 n$ cycles
(n = number of shifts)

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

00	4	$240 + n$
----	---	-----------

18-bit
option

The contents of the A register are rotated left n places ($n = 0$ to 378). Bit position A_{15} (A_{17}) is rotated into bit position A_0 .

Indexing: No
Indirect Addressing: No
Registers Altered: A

LRLB Logical Rotate Left B Timing: $1 + 0.25 n$ cycles
(n = number of shifts)

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

00	4	$040 + n$
----	---	-----------

18-bit
option

The contents of the B register are rotated n positions to the left ($n = 0$ to 378). Bit position B_{15} (B_{17}) is rotated into bit position B_0 .

Indexing: No
Indirect Addressing: No

LLSR Long Logical Shift Right Timing: $1 + 0.50 n$ cycles
(n = number of shifts)

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

00	4	$540 + n$
----	---	-----------

18-bit
option

The contents of the A and B registers are shifted right n positions ($n = 0$ to 378). Bits shifted out of the low-order position of B are lost. Zeros are shifted into the high-order position of the A register.

Indexing: No
Indirect Addressing: No
Registers Altered: A, B

LLRL Long Logical Rotate Left Timing: $1 + 0.50 n$ cycles
(n = number of shifts)

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

00	4	$440 + n$
----	---	-----------

18-bit
option

The contents of the A and B registers are rotated n positions to the left ($n = 0$ to 378). Bit position A_{15} (A_{17}) is shifted into bit position B_0 .

Indexing: No
Indirect Address: No
Registers Altered: A, B

ASRA Arithmetic Shift A Right Timing: $1 + 0.25 n$ cycles
(n = number of shifts)

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

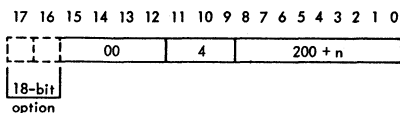
00	4	$300 + n$
----	---	-----------

18-bit
option

The contents of the A register are shifted n positions to the right ($n = 0$ to 378). Bits shifted out of the low-order positions of A are lost. The sign bit of A, A₁₅ (A₁₇) is extended n places to the right.

Indexing: No
Indirect Addressing: No
Registers Altered: A

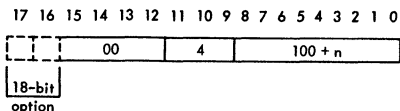
ASLA Arithmetic Shift A Left Timing: $1 + 0.25 n$ cycles
($n = \text{number of shifts}$)



The contents of the A register are shifted n places to the left ($n = 0$ to 378). The sign bit, A₁₅ (A₁₇), is retained and zeros are shifted into the low-order positions of A. Bits shifted out of A₁₄ (A₁₆) are lost.

Indexing: No
Indirect Addressing: No
Registers Altered: A

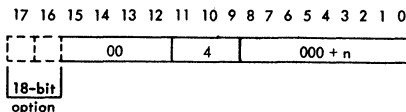
ASRB Arithmetic Shift B Right Timing: $1 + 0.25 n$ cycles
($n = \text{number of shifts}$)



The contents of the B register are shifted n places to the right ($n = 0$ to 378). Information shifted out of the low-order position of B are lost. The sign bit of B, B₁₅ (B₁₇) is extended n places to the right.

Indexing: No
Indirect Addressing: No
Register Altered: B

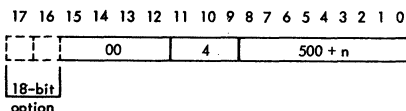
ASLB Arithmetic Shift B Left Timing: $1 + 0.25 n$ cycles
($n = \text{number of shifts}$)



The contents of the B register are shifted n places to the left ($n = 0$ to 378). The sign bit of B, B₁₅ (B₁₇), is retained and zeros are shifted into the low-order positions of B. Bits shifted out of B₁₄ (B₁₆) are lost.

Indexing: No
Indirect Addressing: No

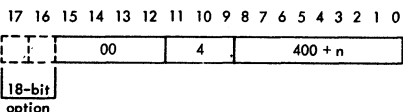
LASR Long Arithmetic Shift Right Timing: $1 + 0.50 n$ cycles
($n = \text{number of shifts}$)



The contents of the A and B registers are shifted n places to the right ($n = 0$ to 378). Bit position A₀ is shifted into bit position B₁₄ (B₁₆). The sign of the A register, A₁₅ (A₁₇), is extended n places to the right. The sign bit, B₁₅ (B₁₇) of the B register remains unchanged. Bits shifted out of the low-order position of the B register are lost.

Indexing: No
Indirect Addressing: No
Register Altered: A, B

LASL Long Arithmetic Shift Left Timing: $1 + 0.50 n$ cycles
($n = \text{number of shifts}$)



The contents of the A and B registers are shifted n places to the left ($n = 0$ to 378). Bit position B14 (B16) is shifted into bit position A0, with the sign of B, B15 (B17) remaining unchanged. The sign of the A register, A15 (A17) is not altered. Information shifted out of A14 (A16) is lost and zeros are shifted into the low-order positions of the B register.

Indexing: No

Indirect Addressing: No

Registers Altered: A, B

Register change group. The register change instruction group provides a macrooperation facility, in that these instructions may combine several register change operations in a

single instruction. The instruction format is shown in figure 3-7.

The memory address field (M) defines the source and destination of a parallel word transfer within the operational register set A, B and X. Any combination of registers may be selected. The M field also specifies whether the word transferred will be unchanged, incremented, decremented, or complemented. The transfer may also be conditional on the overflow indicator.

Table C-6, Appendix C, defines the transfer control specified by the M field. If more than one source register is specified, the result will be the inclusive-OR of the group. Complementing causes transfer of the complement of the inclusive-OR (NOR) of a combination of source registers.

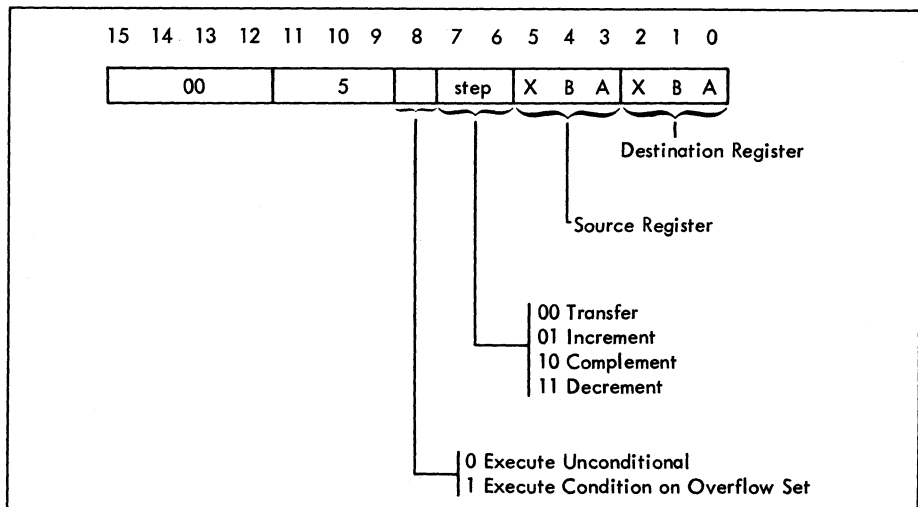


Figure 3-7. Register Change Instruction

A total of 512 different register change operations are possible. The most useful instructions are contained in the mnemonic repertoire recognized by the DAS assembler, summarized in table C-6(b).

IAR Increment A Register Timing: 1 cycle

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

00 5 111

18-bit
option

IBR Increment B Register Timing: 1 cycle

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

00 5 122

18-bit
option

IXR Increment X Register Timing: 1 cycle

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

00 5 144

18-bit
option

The contents of the A (B, X) register are incremented by one, mod 2^{16} (2^{18}). If the sign of the A (B, X) register changes from plus to minus, the overflow indicator (OF) is set.

Indexing: No

Indirect Addressing: No

Registers Altered: A (B, X) OF

DAR Decrement A Register Timing: 1 cycle

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

00 5 311

18-bit
option

DBR Decrement B Register Timing: 1 cycle

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

00 5 322

18-bit
option

DXR Decrement X Register Timing: 1 cycle

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

00 5 344

18-bit
option

The contents of the A (B, X) register are decremented by one, mod 2^{16} (2^{18}). If the sign bit of the A (B, X) register is changed from minus to plus, the overflow indicator (OF) is set.

Indexing: No

Indirect Addressing: No

Registers Altered: A (B, X), OF

CFA Complement A Register Timing: 1 cycle

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

00 5 211

18-bit
option

CPB Complement B Register Timing: 1 cycle

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

00 5 222

18-bit
option

CPX Complement X Register Timing: 1 cycle

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

00 5 244

18-bit
option

The contents of the A (B, X) register are complemented (1's-complement).

Indexing: No
Indirect Addressing: No
Register Altered: A (B, X)

TAB Transfer A Register to B Register Timing: 1 cycle

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

00	5	012
----	---	-----

18-bit
option

The contents of the A register are placed in the B register.

Indexing: No
Indirect Addressing: No
Registers Altered: B

TAX Transfer A Register to X Register Timing: 1 cycle

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

00	5	014
----	---	-----

18-bit
option

The contents of the A register are placed in the X register.

Indexing: No
Indirect Addressing: No
Registers Altered: X

TBA Transfer B Register to A Register Timing: 1 cycle

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

00	5	021
----	---	-----

18-bit
option

The contents of the B register are placed in the A register.

Indexing: No
Indirect Addressing: No
Registers Altered: A

TBX Transfer B Register to X Register Timing: 1 cycle

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

00	5	024
----	---	-----

18-bit
option

The contents of the B register are placed in the X register.

Indexing: No
Indirect Addressing: No
Registers Altered: X

TXA Transfer X Register to A Register Timing: 1 cycle

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

00	5	041
----	---	-----

18-bit
option

The contents of the X register are placed in the A register.

Indexing: No
Indirect Addressing: No
Registers Altered: A

TXB Transfer X Register to B Register Timing: 1 cycle

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

00	5	042
----	---	-----

18-bit
option

The contents of the X register are placed in the B register.

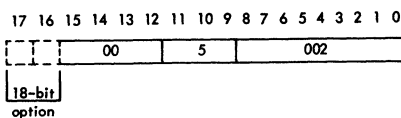
TZA Transfer Zero to A Register Timing: 1 cycle

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

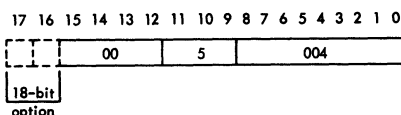
00	5	001
----	---	-----

18-bit
option

TZX Transfer Zero to B Register Timing: 1 cycle



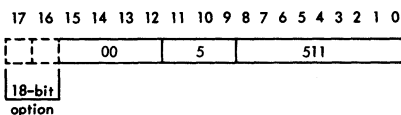
TZX Transfer Zero to X Register Timing: 1 cycle



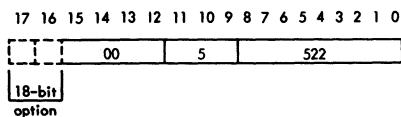
The A (B, X) register is cleared to zero.

Indexing: No
Indirect Addressing: No
Registers Altered: A (B, X)

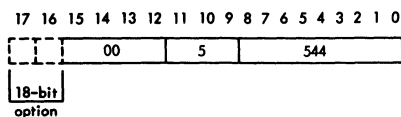
AØFA Add Overflow to A Register Timing: 1 cycle



AØFB Add Overflow to B Register Timing: 1 cycle



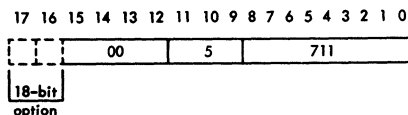
AØFX Add Overflow to X Register Timing: 1 cycle



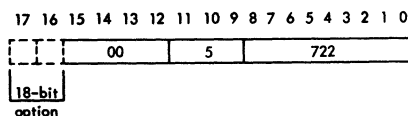
The contents of the overflow indicator (OF) are added to the A (B, X) register, mod 2^{16} (2^{18}). The sum is placed in the A (B, X) register. The overflow flip-flop does not change.

Indexing: No
Indirect Addressing: No
Registers Altered: A (B, X)

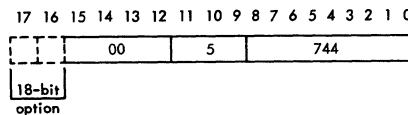
SØFA Subtract Overflow from A Register Timing: 1 cycle



SØFB Subtract Overflow from B Register Timing: 1 cycle



SØFX Subtract Overflow from X Register Timing: 1 cycle



The contents of the overflow indicator (OF) are subtracted from the A (B, X) register, mod 2^{16} (2^{18}). The overflow flip-flop does not change.

Indexing: No
Indirect Addressing: No
Registers Altered: A (B, X)

3.2 DOUBLE-WORD INSTRUCTIONS

Double-word instructions may be either addressing or non-addressing. The instructions of the double-word addressing group are jump, jump and mark, execute and extended addressing.

The instructions in the double-word non-addressing group are the immediate instructions.

Double-Word Addressing Instructions

For double-word addressing instructions, the second word is contained in the memory location following the instruction word. The second word may contain an operand or an address. The address may be either indirect or direct. The general flow chart for double-word instructions is shown in figure 3-8.

Bits 0 through 8 determine the conditions for execution of the instruction. The condition is tested if the corresponding bit is equal to one. For example, if bit 0 equals one, the instruction will examine the status of the overflow flip-flop. If overflow is set, the command will be executed. If overflow is not set, the next instruction in sequence will be executed.

Sequence change instruction group. For this instruction group, the memory address field *M*, contains a set of nine flags which define the logical conditions for execution of the jump function. The jump address is contained in the second word of the double-word instruction.

There are 3 basic instructions in this group, the jump, jump and mark, and execute. They all use the same condition (*M*-field) logic to determine whether or not to execute

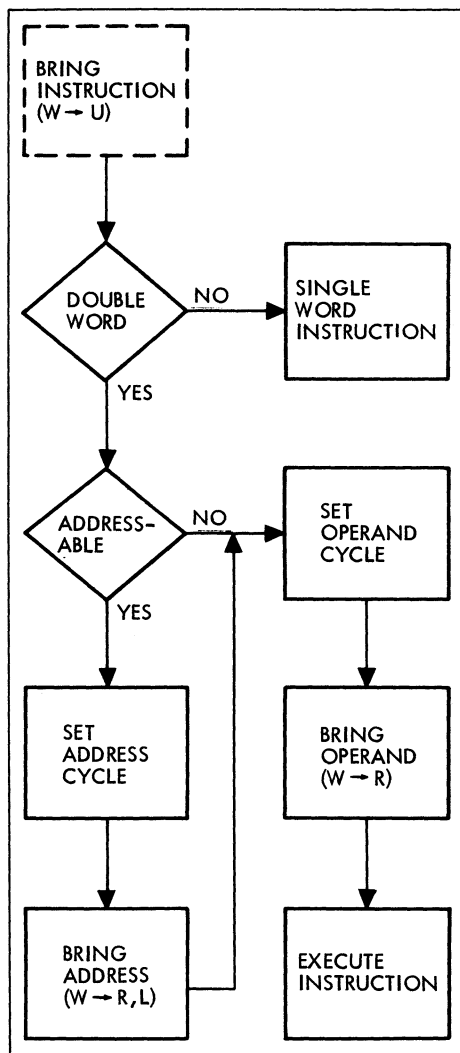
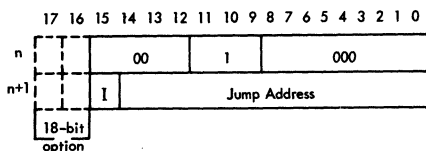


Figure 3-8. Double Word Instruction, General Flow

the sequence change. The condition logic is illustrated in table C-7, Appendix C. The test condition is the logical-AND of all ones in the field. Thus, there are 512 possible combinations, but not all are useful. The most useful conditional instructions are contained in the mnemonic instruction repertoire recognized by the DAS assembler, summarized in table C-7(b). The general flow for the conditional instructions is shown in figure 3-9. The three different sequence change instructions are described individually.

Jump Instruction Group

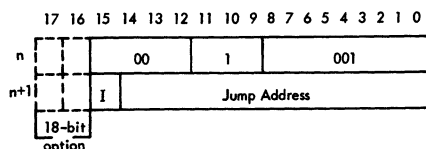
JMP Jump Unconditionally Timing: 2 cycles



The next instruction executed is at the jump address.

Indexing: No
Indirect Addressing: Yes
Registers Altered: P

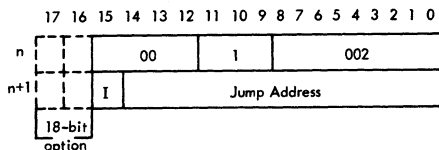
JOF Jump if Overflow Indicator Set Timing: 2 cycles



If the overflow indicator (OF) is set, the next instruction executed is at the jump address. If the overflow indicator is not set, the next instruction in sequence is executed. The overflow indicator is reset upon execution of the JOF instruction.

Indexing: No
Indirect Addressing: Yes
Registers Altered: OF

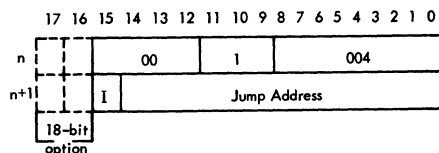
JAP Jump if A Register Positive Timing: 2 cycles



If the contents of the A register are positive or zero, the next instruction executed is at the jump address. If the A register is negative, the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: P

JAN Jump if A Register Negative Timing: 2 cycles



If the A register is negative, the next instruction executed is at the jump address. If the

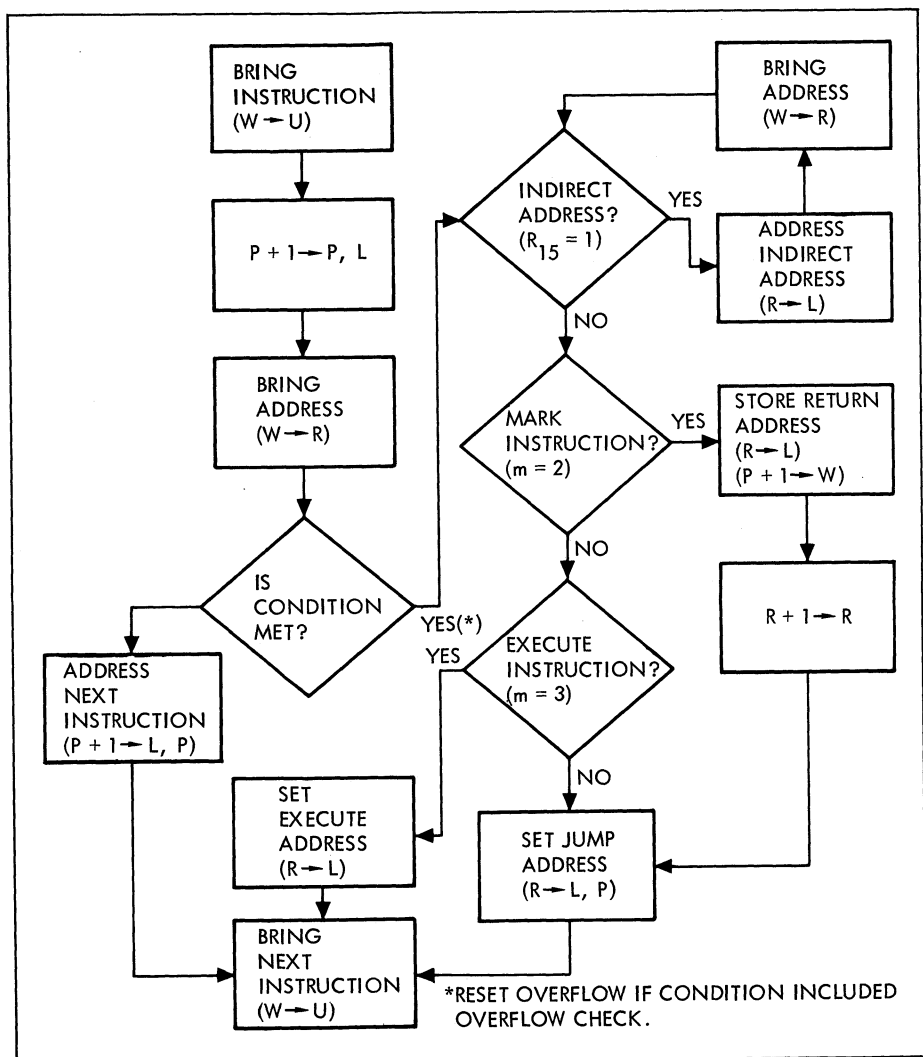
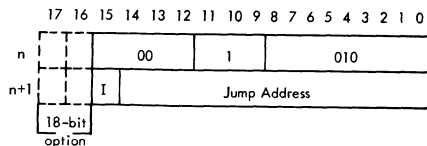


Figure 3-9. Sequence Change Instruction, General Flow

A register is positive, the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: P

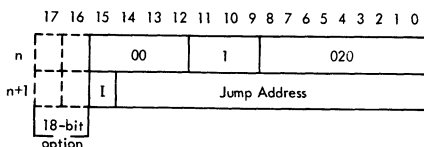
JAZ Jump if A Register Zero Timing: 2 cycles



If the A register is zero, the next instruction executed is at the jump address. If the A register is not zero, the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: P

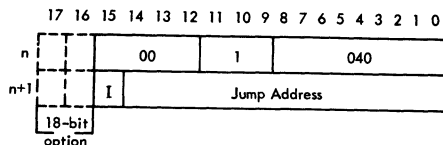
JBZ Jump if B Register Zero Timing: 2 cycles



If the B register is zero, the next instruction executed is at the jump address. If the B register is not zero, the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: P

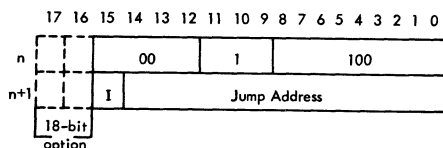
JXZ Jump if X Register Zero Timing: 2 cycles



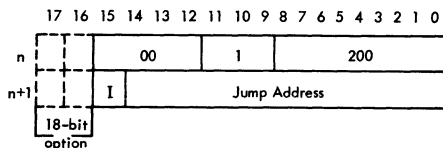
If the index register (X) is zero, the next instruction executed is at the jump address. If the register is not zero, the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: P

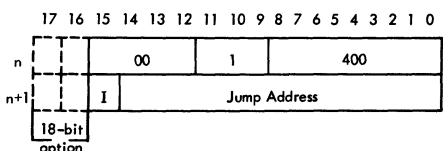
JSS1 Jump if Sense Switch 1 Set Timing: 2 cycles



JSS2 Jump if Sense Switch 2 Set Timing: 2 cycles



JSS3 Jump if Sense Switch 3 Set Timing: 2 cycles



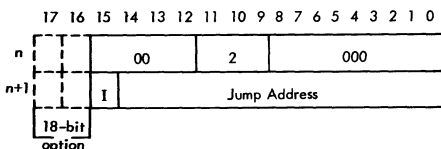
If sense switch 1 (2, 3) is set, the next instruction executed is at the jump address. If

the sense switch being tested is not set, the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: P

Jump-and-Mark Instruction Group

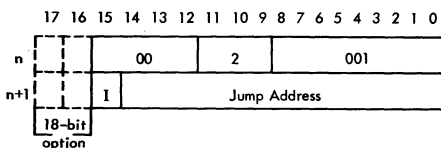
JMPM Jump and Mark Unconditionally Timing: 3 cycles



The contents of the instruction counter (P) are stored at the jump address. The next instruction executed is at the jump address plus one.

Indexing: No
Indirect Addressing: Yes
Registers Altered: Jump address, P

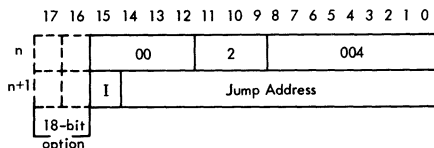
JOFM Jump and Mark if Overflow Set Timing: 2-3 cycles



If the overflow indicator (OF) is set, the contents of the instruction counter (P) are stored at the jump address, and the instruction at the jump address plus one is executed. If the overflow indicator is not set, the next instruction in sequence is executed. The overflow indicator is reset upon execution of the JOFM instruction.

Indexing: No
Indirect Addressing: Yes
Registers Altered: Jump address, P, OF

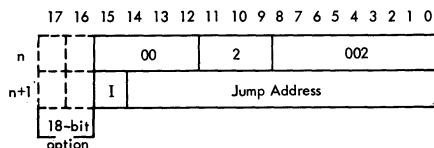
JANM Jump and Mark if A Register Negative Timing: 2-3 cycles



If the A register is negative, the contents of the instruction counter (P) are placed at the jump address, and the instruction at the jump address plus one is executed. If the A register is positive, the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: Jump address, P

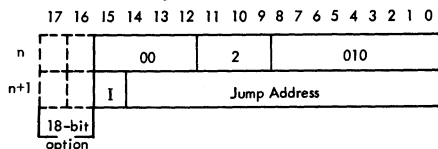
JAPM Jump and Mark if A Register Positive Timing: 2-3 cycles



If the A register is positive or zero, the contents of the instruction counter (P) are placed at the jump address, and the instruction at the jump address plus one is executed. If the A register is negative, the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: Jump address, P

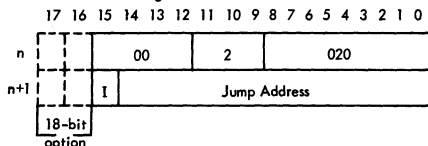
JAZM Jump and Mark if A Register Zero Timing: 2-3 cycles



If the A register is zero, the instruction counter (P) is placed at the jump address and the instruction at the jump address plus one is executed. If the A register is not zero, the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: Jump address, P

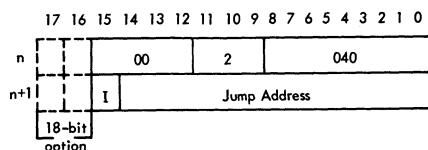
JBZM Jump and Mark if B Register Zero Timing: 2-3 cycles



If the B register is zero, the contents of the instruction counter (P) are placed at the jump address, and the instruction at the jump address plus one is executed. If the B register is not zero, the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: Jump address, P

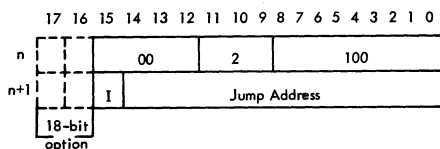
JXZM Jump and Mark if X Register Zero Timing: 2-3 cycles



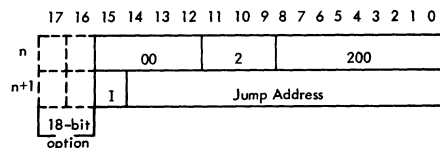
If the X register is zero, the contents of the instruction counter (P) are placed at the jump address and the instruction at the jump address plus one is executed. If the X register is not zero, the next instruction in sequence is executed.

Indexing: No
Indirect Addressing: Yes
Registers Altered: Jump address, P

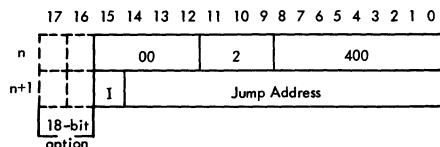
JSTM Jump and Mark if Sense Switch 1 Set Timing: 2-3 cycles



JS2M Jump and Mark if Sense Switch 2 Set Timing: 2-3 cycles



JS3M Jump and Mark if Sense Switch 3 Set Timing: 2-3 cycles



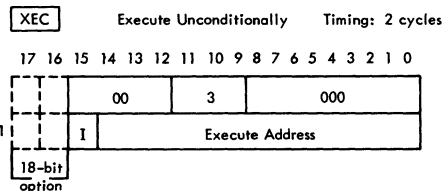
If sense switch 1 (2, 3) is set, the instruction counter (P), is placed at the jump address, and the instruction at the jump address plus one is executed. If the tested SENSE switch is not set, the next instruction in sequence is executed.

Indexing: No
 Indirect Addressing: Yes
 Registers Altered: Jump address, P

Execute Instruction Group

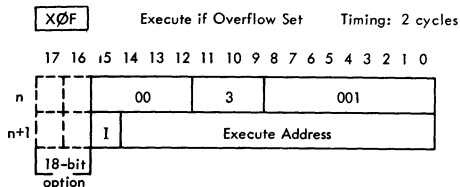
It is important to note that only single-word instructions should be Executed. The single-word instruction groups are load/store, arithmetic, logical, control, shift and register change. Note that any addressing relative to the P register is relative to the normal sequence and not relative to the executed instruction.

If the execute is attempted on double-word instructions, erroneous operations will occur. The P register is incremented during the execution of double-word instructions, and the return from execute will be to the wrong place. The double-word instruction groups are jump, jump and mark, execute, extended addressing (optional) and immediate.



The instruction located at the execute address is executed and then the next instruction in sequence is executed.

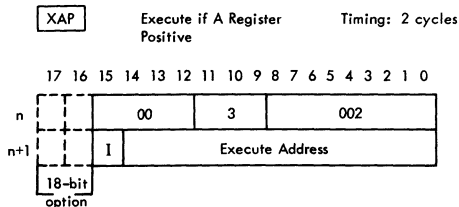
Indexing: No
 Indirect Addressing: Yes
 Registers Altered: None



If the overflow indicator (OF) is set, the instruction at the execute address is executed, and then the next instruction in sequence is executed.

If the overflow indicator is not set, the next instruction in sequence is executed. Execution of the XOF instruction resets the overflow indicator.

Indexing: No
 Indirect Addressing: Yes
 Registers Altered: OF (reset)

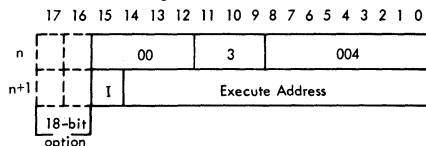


If the A register is positive or zero, the instruction at execute address is executed, and then the next instruction in sequence is executed. If the A register is negative, the next instruction in sequence is executed.

Indexing: No
 Indirect Addressing: Yes
 Registers Altered: None

XANExecute if A Register
Negative

Timing: 2 cycles



If the A register is negative, the instruction at the execute address is executed and then the next instruction in sequence is executed. If the A register is positive, the next instruction in sequence is executed.

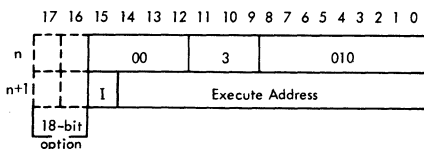
Indexing: No

Indirect Addressing: Yes

Registers Altered: None

XAZExecute if A Register
Zero

Timing: 2 cycles



If the A register is zero, the instruction at the execute address is executed and then the next instruction sequence is executed.

If the A register is not zero the next instruction in sequence is executed.

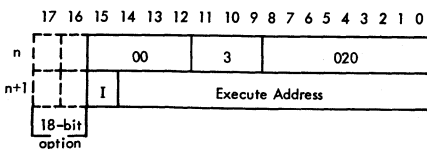
Indexing: No

Indirect Addressing: Yes

Registers Altered: None

XBZExecute if B Register
Zero

Timing: 2 cycles



If the B register is zero, the instruction at the execute address is executed and then the next instruction in sequence is executed.

If the B register is not zero, the next instruction in sequence is executed.

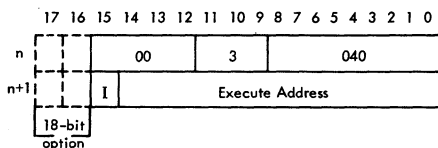
Indexing: No

Indirect Addressing: Yes

Registers Altered: None

XXZExecute if X Register
Zero

Timing: 2 cycles



If the index register (X) is zero, the instruction at the execute address is executed and then the next instruction in sequence is executed:

If the index register is not zero, the next instruction in sequence is executed.

Indexing: No

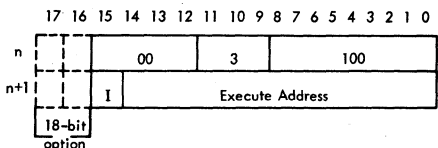
Indirect Addressing: Yes

Register Altered: None

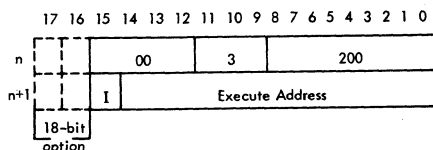
XS1

Execute if Sense Switch 1

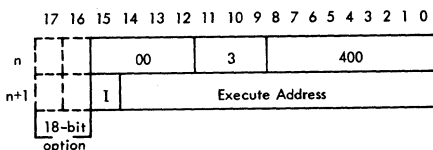
Timing: 2 cycles



XS2 Execute if Sense Switch 2 Timing: 2 cycles



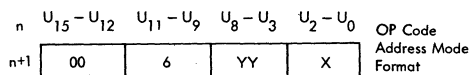
XS3 Execute if Sense Switch 3 Timing: 2 cycles



If sense switch 1, (2, 3) is set, the instruction at the execute address is executed and then the next instruction in the sequence is executed. If the sense switch tested is not set, the next instruction is executed.

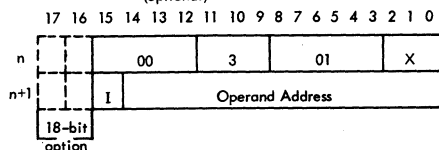
Indexing: No
Indirect Addressing: Yes
Register Altered: None

Extended-addressing instruction group (optional). The extended address mode instructions are similar in format to the immediate instructions. However, the second word of the double-word instruction contains the effective address. The address can be indirect or direct. This is determined by bit 15 of the second word. These instructions are summarized in table C-8, Appendix C.



YY corresponds to the op code and x to the m field of the single-word addressing instructions.

LDAE Load A Register Extended Timing: 3 cycles (optional)

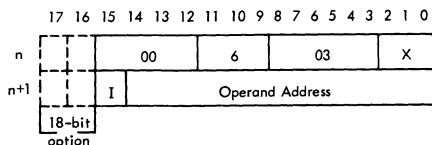


If X =	Address Mode	Effective Address
0-3	Immediate	Second word contains operand
4	Relative to P	Contents of second word plus (P register plus 1)
5	Indexed with X	Contents of second word plus X register
6	Indexed with B	Contents of second word plus B register
7	Direct or indirect	Contents of second word is the direct address if bit 15 is zero. Contents of second word is an indirect address if bit 15 is one.

The contents of the memory location as addressed by the operand address at location $n + 1$ are placed in the A register.

Indexing: Yes
Indirect Addressing: Yes
Register Altered: A

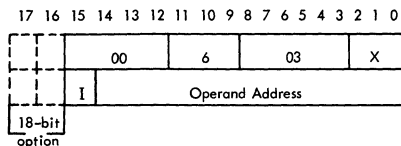
LDBE Load B Register Extended (optional) Timing: 3 cycles



The contents of the memory location as addressed by the operand address at location $n+1$ are placed in the B register.

Indexing: Yes
Indirect Addressing: Yes
Register Altered: B

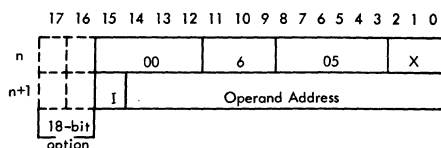
LDXE Load X Register Extended (optional) Timing: 3 cycles



The contents of the memory location as addressed by the operand address at location $n + 1$ are placed in the X register.

Indexing: Yes
Indirect Addressing: Yes
Register Altered: X

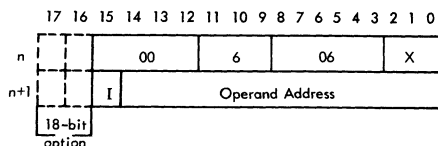
STAE Store A Register Extended (optional) Timing: 3 cycles



The contents of the A register are stored in the memory location as addressed by the operand address at location $n + 1$.

Indexing: Yes
Indirect Addressing: Yes
Register Altered: Memory

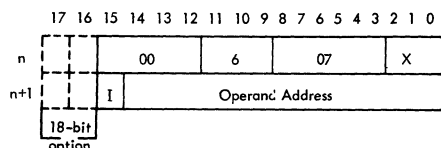
STBE Store B Register Extended (optional) Timing: 3 cycles



The contents of the B register are stored in the memory location as addressed by the operand address to location $n + 1$.

Indexing: Yes
Indirect Addressing: Yes
Register Altered: Memory

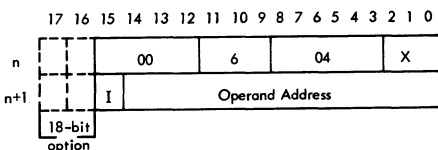
STXE Store Index Register Extended (optional) Timing: 3 cycles



The contents of the index register are stored in the memory location as addressed by the operand address at location $n + 1$.

Indexing: Yes
Indirect Addressing: Yes
Register Altered: Memory

INRE Increment Memory and Replace Extended (optional) Timing: 4 cycles

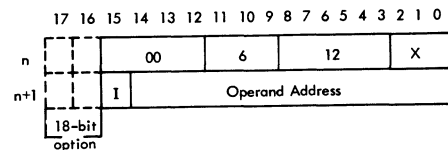


The contents of the memory location as addressed by the operand address at location $n + 1$ are incremented by one, mod 2^{16} (2^{18}).

After execution, if $(M) \geq 2^{15}$ (2^{17}), the overflow indicator (OF) is set.

Indexing: Yes
Indirect Addressing: Yes
Register Altered: Memory, OF

ADDE Add Memory to A Extended (optional) Timing: 3 cycles

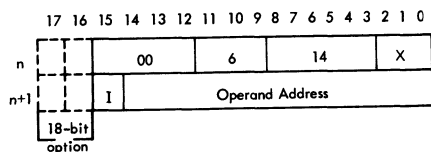


The contents of the memory location as addressed by the operand address at location $n + 1$ are added to the contents of the A register and the sum is placed in the A register.

After execution, if $(A) \geq 2^{15}$ (2^{17}) or $(A) < -2^{15}$ (-2^{17}), the overflow indicator (OF) is set.

Indexing: Yes
Indirect Addressing: Yes
Register Altered: A, OF

SUBE Subtract Memory from A Extended (optional) Timing: 3 cycles

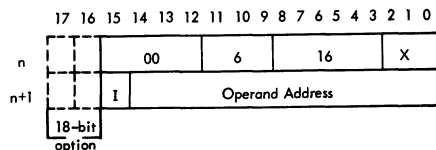


The contents of the memory location as addressed by the operand address at location $n + 1$ are subtracted from the contents of the A register and the difference is placed in the A register.

After execution, if $(A) \geq 2^{15}$ (2^{17}) or $(A) < -2^{15}$ (-2^{17}), the overflow indicator (OF) is set.

Indexing: Yes
Indirect Addressing: Yes
Register Altered: A, OF

MULE Multiply Extended (optional) Timing: 11 cycles (16 bits)
12 cycles (18 bits)



The contents of the B register are multiplied by the contents of the memory location as addressed by the operand address in location

$n + 1$. The original contents of the A register are added to the final product. The product is placed in the A and B registers with the most-significant half of the product in the A register and the least-significant half in the B register. The sign of the product is contained in the sign position of the A register. The sign position of the B register is reset to zero.

The algorithm is in the form

$$(M) \cdot (B) + (A) \rightarrow (A, B)$$

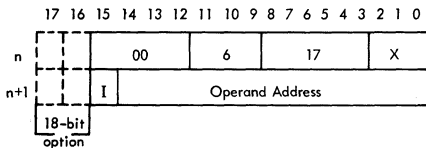
Indexing: Yes

Indirect Addressing: Yes

Register Altered: A, B, OF

DIVE

Divide Extended Timing: 11-15 cycles (16 bits)
(optional) 12-16 cycles (18 bits)



The contents of the A and B registers are divided by the contents of the memory location as addressed by the operand address at location $n + 1$. The quotient is placed in the B register and the remainder is placed in the A register.

The algorithm is in the form:

$$(A, B) / M \begin{cases} (B) \text{ Quotient} \\ (A) \text{ Remainder} \end{cases}$$

$$\text{If } |(A, B) / (M)| < 1$$

(|divisor| > |dividend|, taken as a binary fraction), overflow will not occur. If overflow does occur, the overflow indicator (OF) is set.

Indexing: Yes

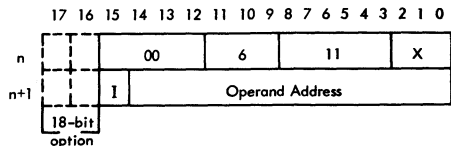
Indirect Addressing: Yes

Register Altered: A, B, OF

ØRAE

Inclusive-OR Memory and A Extended (optional)

Timing: 3 cycles



The inclusive-OR operation is performed between the contents of the A register and the contents of the memory location as addressed by the operand address in location $n + 1$.

The result is placed in the A register. If either the memory or A contains a one in the same position, a one is placed in the result. The truth table is shown below, where $n = \text{bit position}$.

Condition		Result
$A_{(n)}$	Effective Memory Location (n)	$A_{(n)}$
0	0	0
0	1	1
1	0	1
1	1	1

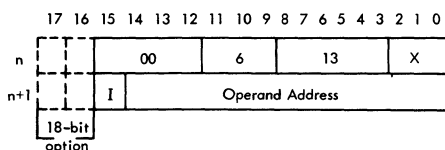
Indexing: Yes

Indirect Addressing: Yes

Register Altered: A

ERAEExclusive-OR Memory
and A Extended (optional)

Timing: 3 cycles



An exclusive-OR operation is performed between the contents of the A register and the contents of the memory location as addressed by the operand address in location $n + 1$. The result is placed in the A register. If the same bit position of the memory location and the A register contains a zero, or if both bit positions contains a one, the result is zero. The truth table is shown below, where n = bit position:

Condition		Result
$A(n)$	Effective Memory Location (n)	$A(n)$
0	0	0
0	1	1
1	0	1
1	1	0

Indexing: Yes

Indirect Addressing: Yes

Register Altered: A

The logical-AND operation is performed between the contents of the A register and the contents of the memory location as addressed by the operand address in location $n + 1$. The result is placed in the A register. If the same bit position of both the memory location and the A register contains a one the result is a one. The truth table is shown below, where n = bit position:

Condition		Result
$A(n)$	Effective Memory Location (n)	$A(n)$
0	0	0
0	1	0
1	0	0
1	1	1

Indexing: Yes

Indirect Addressing: Yes

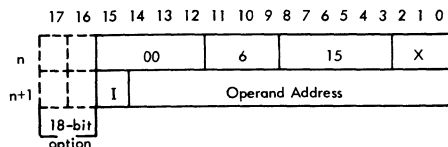
Register Altered: A

Double-Word Non-Addressing Instructions

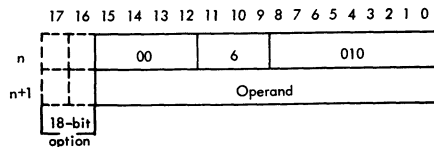
The double-word non-addressing instructions consist of the immediate instruction group. The operand for the immediate instruction is contained in the second word of the double-word instruction. Address modification is not permitted for this group of instructions. The immediate instruction group codes are summarized in table C-9 of Appendix C.

ANAEAND Memory and A
Extended (optional)

Timing: 3 cycles

**LDAI**Load A Register
Immediate

Timing: 2 cycles



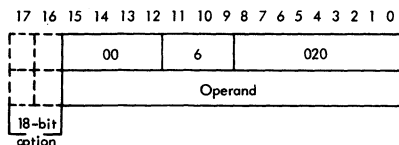
The contents of the operand are placed in the A register.

Indexing: No

Indirect Addressing: No

Registers Altered: A

LDBI Load B Register Immediate Timing: 2 cycles



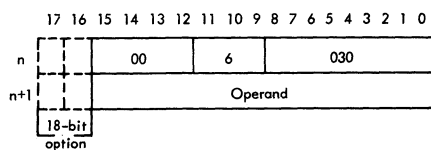
The contents of the operand are placed in the B register.

Indexing: No

Indirect Addressing: No

Registers Altered: B

LDXI Load X Register Immediate Timing: 2 cycles



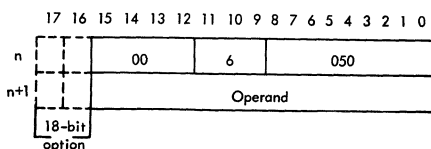
The contents of the operand are placed in the X register.

Indexing: No

Indirect Addressing: No

Registers Altered: X

STAI Store A Register Immediate Timing: 2 cycles



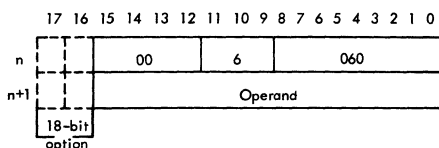
The contents of the A register are placed in the operand location.

Indexing: No

Indirect Addressing: No

Registers Altered: Operand

STBI Store B Register Immediate Timing: 2 cycles



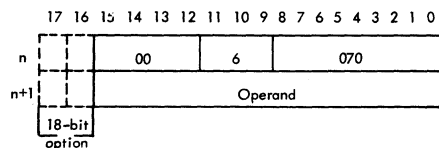
The contents of the B register are placed in the operand location.

Indexing: No

Indirect Addressing: No

Registers Altered: Operand

STXI Store X Register Immediate Timing: 2 cycles



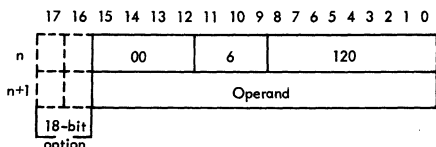
The contents of the index register are placed in the operand location.

Indexing: No

Indirect Addressing: No

Registers Altered: Operand

ADDI Add Immediate Timing: 2 cycles



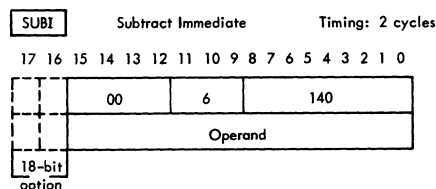
The contents of the A register are added to the contents of the operand. The sum is placed in the A register.

After execution, if $(A) \geq 2^{15} (2^{17})$ or $(A) < -2^{15} (-2^{17})$, the overflow indicator (OF) is set.

Indexing: No

Indirect Addressing: No

Registers Altered: A, OF



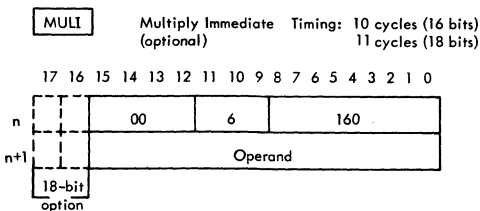
The contents of the operand are subtracted from the contents of the A register. The difference is placed in the A register.

After execution, if $(A) \geq 2^{15} (2^{17})$ or $(A) < -2^{15} (-2^{17})$, the overflow indicator (OF) is set.

Indexing: No

Indirect Addressing: No

Registers Altered: A, OF



The contents of the B register are multiplied by contents of the operand. The original contents of the A register are added to the final product. The product is placed in the A and B registers, with the most-significant half of the product in the A register and the least-significant half in the B register. The sign of the product is contained in the sign position of the A register. The sign position of the B register is reset to zero.

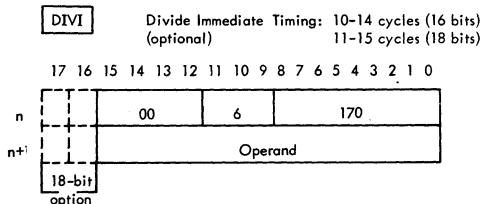
The algorithms is in the form

$$(M) \cdot (B) + (A) \rightarrow (A, B)$$

Indexing: No

Indirect Addressing: No

Registers Altered: A, B, OF



The contents of the A and B registers are divided by the contents of the operand. The quotient is placed in the B register with sign, and the remainder is placed in the A register with the sign of the dividend.

The algorithm is in the form:

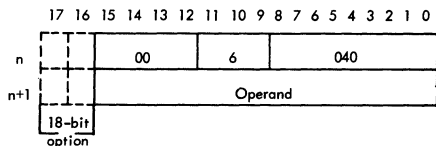
$$(A, B) / (M) \begin{cases} \rightarrow (B) \text{ Quotient} \\ \rightarrow (A) \text{ Remainder} \end{cases}$$

$$\text{If } |(A, B) / (M)| < 1$$

(|divisor| > |dividend|, taken as a binary fraction), overflow will not occur. If overflow does occur, the overflow indicator (OF) is set.

Indexing: No
Indirect Addressing: No
Registers Altered: A, B, OF

INRI Increment and Replace Immediate Timing: 3 cycles

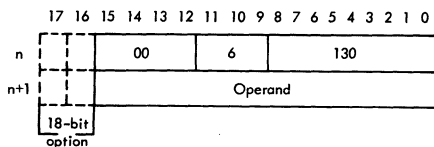


The value of the operand (the second word of the instruction) is incremented by one, mod 2^{16} (2^{18}).

After execution, if the operand $\geq 2^{15}$ (2^{17}), the overflow indicator (OF) is set.

Indexing: No
Indirect Addressing: No
Registers Altered: Operand, OF

ERAI Exclusive-OR Immediate Timing: 2 cycles

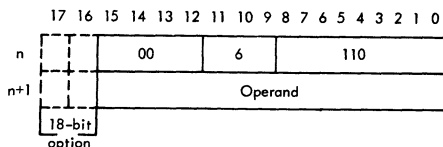


An exclusive-OR is performed between the contents of the operand and the contents of the A register, and the result is placed in the A register. If the same bit position of the operand and the A register contains a zero, or if both bit positions contain a one, the result is zero. The truth table is shown below, where n = bit position.

Condition		Result
$A_{(n)}$	Operand (n)	$A_{(n)}$
0	0	0
0	1	1
1	0	1
1	1	0

Indexing: No
Indirect Addressing: No
Registers Altered: A

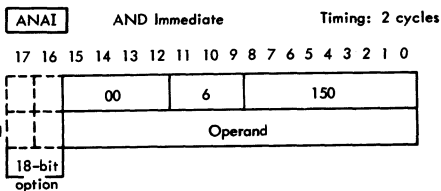
ORAI Inclusive-OR Immediate Timing: 2 cycles



An inclusive-OR is performed between the contents of the operand and the contents of the A register. The result is placed in the A register. If either the operand or the A register contains a one in the same bit position, a one is placed in the result in the A register. The truth table is shown below, where n = bit position:

Condition		Result
$A_{(n)}$	Operand (n)	$A_{(n)}$
0	0	0
0	1	1
1	0	1
1	1	1

Indexing: No
 Indirect Addressing: No
 Registers Altered: A



A logical-AND is performed between the contents of the operand and the contents of the A register. The result is placed in the A register. If the same bit position of the operand and the A register contains a one, the

result is one; otherwise, the result is zero. The truth table is shown below, where n = bit position.

Condition		Result
$A_{(n)}$	Operand (n)	$A_{(n)}$
0	0	0
0	1	0
1	0	1
1	1	1

Indexing: No
 Indirect Addressing: No
 Registers Altered: A

SECTION 4

INPUT/ OUTPUT OPERATION

This section describes the operation and instruction repertoire of the DATA 620/i input/output (I/O) system. The standard computer is equipped with a party line I/O system that has capabilities, under program control, to input data, output data, sense external signals and generate control signals. The DATA 620/i input/output system is designed to facilitate integration of the computer into an overall system. Refer to Sections 6-8, for detailed information required for special interface designs.

A wide selection of peripheral devices can be controlled by the 620/i.

4.1 ORGANIZATION

As shown in the block diagram, figure A-1, Appendix A, the I/O section of the computer communicates with the operational registers and the memory through the internal C bus. Data and control signals are transmitted to and from external peripheral devices through the I/O bus.

Overall Operation

The overall organization of the DATA 620/i I/O system, including a typical set of peripheral devices, is shown in figure A-2. Standard or special peripheral devices are in parallel on the I/O bus.

The following types of I/O commands can be executed by the standard computer.

Single word to/from memory. A single word may be transferred to or from any memory location.

Single word transfer to/from A or B register. A single word may be transferred to or from the A or B register under program control.

Test external sense line. The computer can sense the status of a selected external line under program control.

Generate external control line. An external control code may be transmitted, under program control, from the computer to an external device.

Input/Output Bus Structure

The basic DATA 620/i computer (620/i-00, 622/i-00) is equipped with a positive-voltage-level party-line I/O bus. The party line is a bi-directional common communication channel containing the data and control lines required for system communication. Each transmission on the party line has two phases: The first phase is the route set-up (i.e., device selection); the second is the data transmission.

The party line permits plug-in expansion of all peripheral devices. The party line contains line drivers and line receivers to service up to ten standard peripheral devices. Modifications to the computer are not required to add peripherals. Each standard peripheral device contains a party-line data buffer. Thus, no device can tie-up the party

line. The party line technique solves the troublesome problems usually encountered with on-site system expansion.

Input/Output Operations

During information transfers over the I/O bus, the E-bus lines may carry control codes, addresses, or data, depending upon the type of operation being performed. Table 4-1 defines the I/O cable control signals used to synchronize all I/O operations. Table 4-2 summarizes the signals on the interrupt cable.

NOTE

An I/O command is not transmitted intact over the E bus. Bits 11-15 are decoded in the central processor. The processor then generates an E-bus bit (EB11-EB15). Only one of these bits is true for each type of command. Bits 0-8 of the command are transmitted unchanged on the I/O cable.

4.2 PROGRAM CONTROL FUNCTIONS

Interfacing functions fall into two major categories: programmed operations and automatic operations. The programmed operations are: external control (single-bit out), sense operations (testing a single bit), data transfer in (full-word input) and data transfer out (full-word output). The following paragraphs describe the programmed operations and examples of their use. The I/O instruction group is summarized in table C-10 of Appendix C. This group of instructions is standard for the DATA 620/i.

External Control

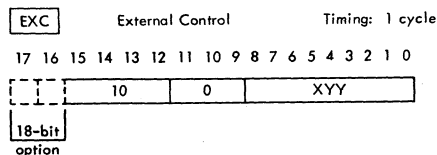
The external control instruction is a single word, non-addressing instruction. It places a function code, contained in bits 0-8, on the E bus to initiate a control operation in an external device.

Table 4-1. I/O-Cable Control Line Signals

Control Line	Signal Name	Function
Function Ready	FRYX-I	Indicates that the E bus contains control or address information.
Data Ready	DRYX-I	Indicates that the E bus contains data.
Sense Response	SERX-I	Indicates logical state of line queried by sense line address on E bus.
Interrupt Acknowledge	IUAX-I	Indicates that external interrupt or trap demand is being acknowledged. Address is placed on E bus and removed with the function-ready signal.
System Reset	SYRT-I	Reset line for initializing peripheral controllers. Energized by console RESET switch.

Table 4-2. Interrupt-Cable Control Line Signals

Control Line	Signal Name	Function
Interrupt Request	IURX-I	Indicates a demand from the Interrupt module to force program to take one instruction from location specified by address on E bus. This address will be placed on E bus when IUAX-I is true.
Trap-Out Request	TPOX-I	Indicates that a buffer interlace controller or other trap device is requesting data transfer from memory.
Trap-In Request	TPIX-I	Indicates that a buffer interlace controller or other trap device is requesting data transfer to memory.
Interrupt Clock	IUXC-I	1.1-MHz clock provided on cable for interrupt module. May be used in any interface design. This clock is not present if the direct-memory-access-and-interrupt option is not included in the system.
Priority Out	PRIX-I	Priority lines used with interrupt and buffer-interlace-controller modules for priority determination.
Priority In	PR4X-I	Priority line returned to computer for permitting console interrupt.
Priority 2 and 3	PR2X-I, PR3X-I	Intermediate priority lines that are used to assign priority positions among trap and interrupt devices.
Interrupt Jump	IUJP-I	Indicates that instruction at interrupt location is a jump-and-mark (two-word) instruction.



function to be performed by the selected device is contained in the X portion.

Indexing: No
 Indirect Addressing: No
 Registers Altered: None

Program Sense

The sense instruction is a double-word, addressing instruction that senses the logical state of an external line. Figure 4-1 shows the execution of this instruction.

The nine bits represented by XXX are placed on the E bus for transmission to the peripheral controllers. The device address is contained in the YY portion of the data, and the

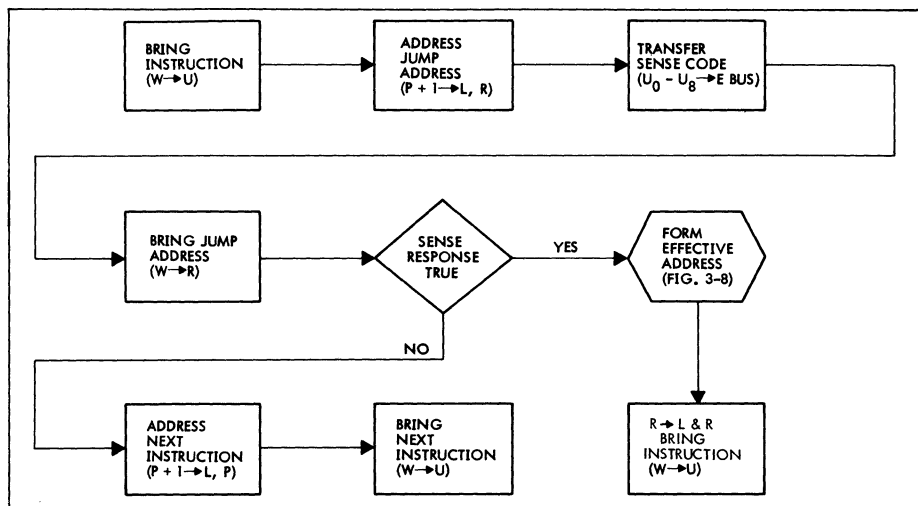
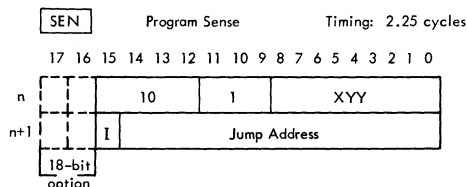


Figure 4-1. Sense Instruction, General Flow



I = 0, word contains an address

I = 1, word contains an indirect address

The nine bits represented by XYY are placed on the party line I/O bus and represent the condition to be tested. X defines a specific line within device YY. The associated peripheral controller replies with a true or false signal.

If a true signal is received by the DATA 620/i, a jump is made to the jump address.

If a false signal is received, the next instruction in sequence is executed.

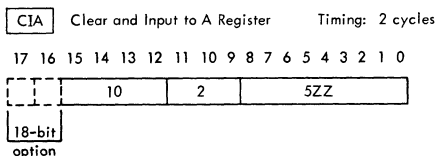
Indexing: No

Indirect Addressing: Yes

Registers Altered: P

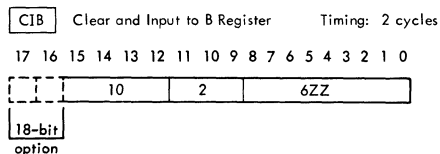
Data Transfer In

Two types of data transfer in instructions are provided: input to operational registers, and input directly to memory. The first type of input instruction is a single-word, non-addressing instruction; the second type is a double-word addressing instruction.



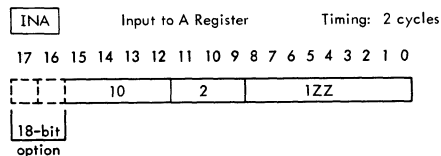
The A register is cleared and a data word from the selected device, ZZ, is transferred to the A register.

Indexing: No
Indirect Addressing: No
Registers Altered: A



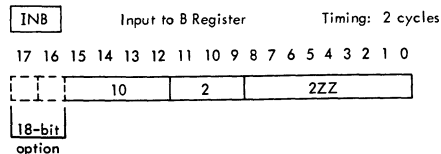
The B register is cleared and a data word from the selected device, ZZ, is transferred to the B register.

Indexing: No
Indirect Addressing: No
Registers Altered: B



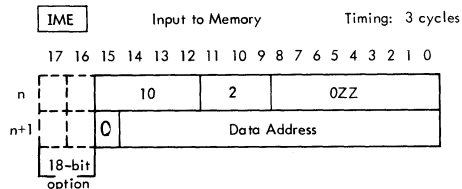
A data word from the selected device, ZZ, is inclusively-OR ed with the contents of the A register.

Indexing: No
Indirect Addressing: No
Registers Altered: A



A data word from the selected device, ZZ, is inclusively-OR ed with the contents of the B register.

Indexing: No
Indirect Addressing: No
Registers Altered: B

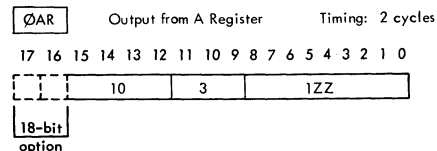


A data word from the selected device, ZZ, is placed in the cleared effective memory address. Figure 4-2 shows the execution of this instruction.

Indexing: No
Indirect Addressing: No
Registers Altered: Memory

Data Transfer Out

Two types of output data transfer instructions are provided: output from operational registers and output from memory. The first type of instruction is a single-word, non-addressing instruction; the second type is a double-word, addressing instruction.



The contents of the A register are transferred to the selected device, ZZ.

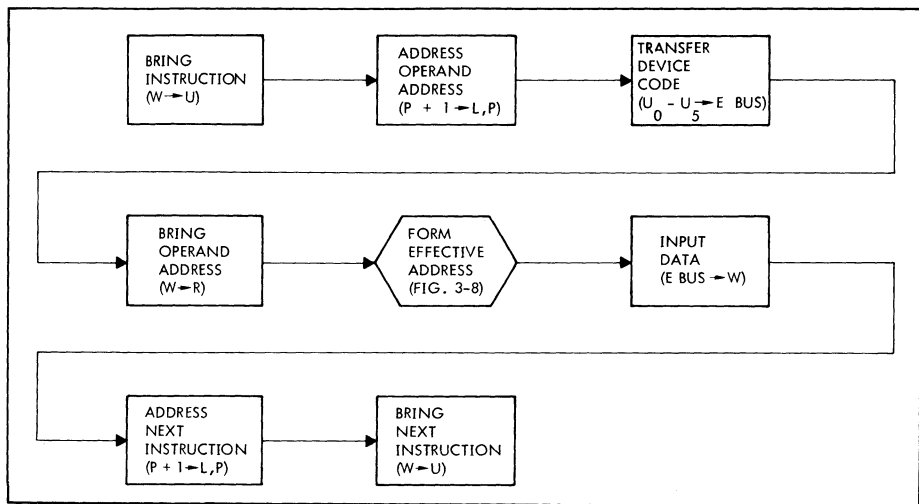
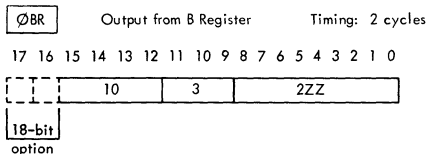


Figure 4-2. Input-to-Memory, General Flow

Indexing: No
Indirect Addressing: No
Registers Altered: None

The contents of the effective memory location are transferred to the selected device, ZZ.



The contents of the B registers are transferred to the selected device, ZZ.

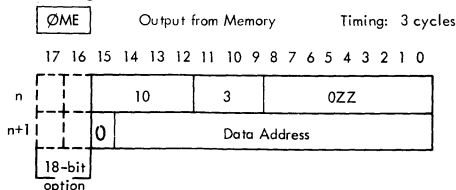
Indexing: No
Indirect Addressing: No
Registers Altered: None

Indexing: No
Indirect Addressing: No
Registers Altered: None

4.3 OPTIONAL AUTOMATIC CONTROL FUNCTIONS (direct-memory-access-and-interrupt logic option)

Two types of computer timing sequences are provided to automatically transfer control and information signals between peripheral devices and the DATA 620/i:

- An interrupt timing sequence is initiated when the DATA 620/i recognizes an external interrupt signal. This sequence forces the computer to execute an instruction



at the memory location specified by interrupt logic through the E bus.

- b. A trap timing sequence is initiated when an external device signals that it must transfer a word to or from memory. The external device must supply the memory address of the word through the E bus. This sequence delays the internal program sequence for the time required to execute the I/O transfer (2.7 microseconds).

The devices that demand either of those automatic sequences must first have priorities to resolve two or more simultaneous demands for service. The priorities of devices demanding service are determined every 0.9 microseconds, and are clocked by the interrupt clock. Refer to Sections 6 - 8 for a more detailed description. Priority assignment for devices on the I/O cable is optional and is a part of the system definition. Priorities may be fixed for any given configuration by properly connecting priority lines in the I/O cable. Priorities can be altered if the definition changes.

Interlace Data Transfers

Interlace optional data transfers may be performed concurrently with internal program operation. This type of operation uses the computer trap-timing sequence to delay the program for 2.7 microseconds while a word is transferred between memory and a peripheral device. The transfer is controlled by the external device, which must transmit the memory address of the data word, and must synchronize the operation using the signals transmitted on the I/O control lines. The maximum interlace transfer rate is 202,000 words per second.

The general trap-sequence flow is shown in figure 4-3. The maximum computer delay in

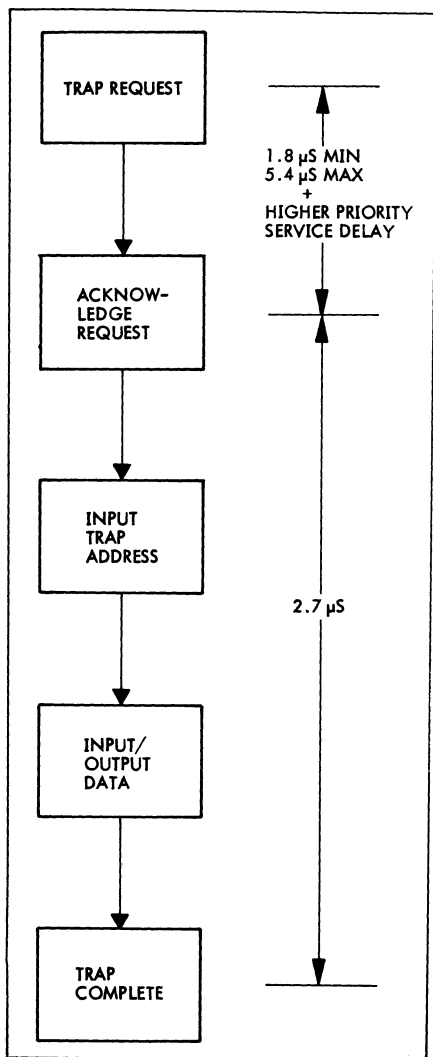


Figure 4-3. Trap Sequence, General Flow

acknowledging a trap request is 5.4 microseconds. However, the time delay experienced by a specific controller in receiving acknowledgment to a trap request may be extended by the time required for the computer to service higher-priority requests.

Special peripheral controllers designed for system applications (such as A/D and D/A converters) may utilize the trap facilities of the computer to implement automatic I/O operations (refer to Sections 6 - 8 detailed design information). A buffer interlace controller (BIC) is also available for use with all standard DATA 620/i peripheral equipment. Special system devices may be interfaced for interlace operations under control of the BIC.

Program Interrupt (optional)

The DATA 620/i has a multi-level interrupt system with single execute, on/off and selective arm/disarm capability. Each interrupt line is assigned a unique memory interrupt address which is the first of a pair of locations. The system is modular and expandable in sets of eight levels.

Each optional interrupt line has an enable/disable flip-flop which is addressable and set by interrupt control instructions. If signals exist on one or more interrupt lines, the

highest-priority line is recognized and the corresponding memory destination address is transmitted to the DATA 620/i after the current instruction is executed.

For each group of interrupts, enable is determined by an 8-bit mask word transferred under program control to the arm/disarm flip-flops in the interrupt system. The action initiated by an interrupt subroutine causes the interrupting device to remove its request signal. An acknowledgment of an interrupt causes the instruction located at the interrupt address to be executed. The instruction can be any of the DATA 620/i repertoire. This technique permits the interrupts to be of the single-execute type, whereby single-instruction responses to external signals can be serviced in one instruction period. A real-time clock can be implemented with an interrupt line and an external pulse generator. An automatic data channel can be implemented with as few as two interrupt lines. If the executed instruction is a jump-and-mark instruction, the interrupt system is automatically inhibited, permitting the inhibit to be terminated under program control. While in the inhibit mode, the interrupt subroutine may selectively enable and disable interrupt levels, and then enable the system, permitting the selected levels to interrupt the level being processed.

SECTION 5

CONTROL CONSOLE OPERATION

5.1 CONTROLS AND INDICATORS

The DATA 620/i console (figure 5-1) provides controls and displays required for operator communication with the computer. The contents of all operational registers, including the instruction register, can be displayed in binary-octal form. During normal operation (run mode) the contents of the computer C bus are displayed continuously. Data entry into a selected operational register is accomplished in the step mode (computer halted) by momentary-contact switches. During the run mode, these switches are inhibited to prevent accidental alteration of the register contents.

Control switches allow the operator to manually alter normal program operation. These switches, described in table 5-1, provide considerable control flexibility and are useful for maintenance, troubleshooting and program debugging. The sense switches are useful in normal program operation to allow selection of particular program sequences to be executed.

5.2 MANUAL OPERATION

Control console operation may be understood by reference to table 5-1 and figure 5-1. The following paragraphs describe typical operating sequences which illustrate normal use of the computer.

Power Control

The POWER switch applies power to computer logic memory, and controller logic.

Manual Program Entry and Execution

When the computer is halted (step mode), programs and data may be read from memory, and entered into memory, and a pre-stored program may be manually executed.

To load words into memory (either instructions or data), set the desired word in the A, B or X register. Set the appropriate store-type instruction (STA, STB, STX) with the desired operand address in the instruction (U) register; then press the STEP switch to execute the store operation.

To display the contents of any memory cell in the A, B or X register; set the appropriate load-type instruction (LDA, LDB, LDX) with the proper memory address in the instruction register; then press the STEP switch to load the selected word into the register. To manually execute a program stored in memory, set the starting address of the program in the program counter. When the STEP switch is pressed, the instruction contained in the instruction register is executed, and the instruction of the selected address is transferred to the instruction register. Repeated operation of the STEP switch will then step through the program one instruction at a time. All operations such as multi-level indirect addressing will be performed for each instruction as the STEP switch is operated. Note that I/O instructions involving an asynchronous device that transfers data in a block (such as a magnetic tape unit or teletype) generally cannot be operated in the step mode.

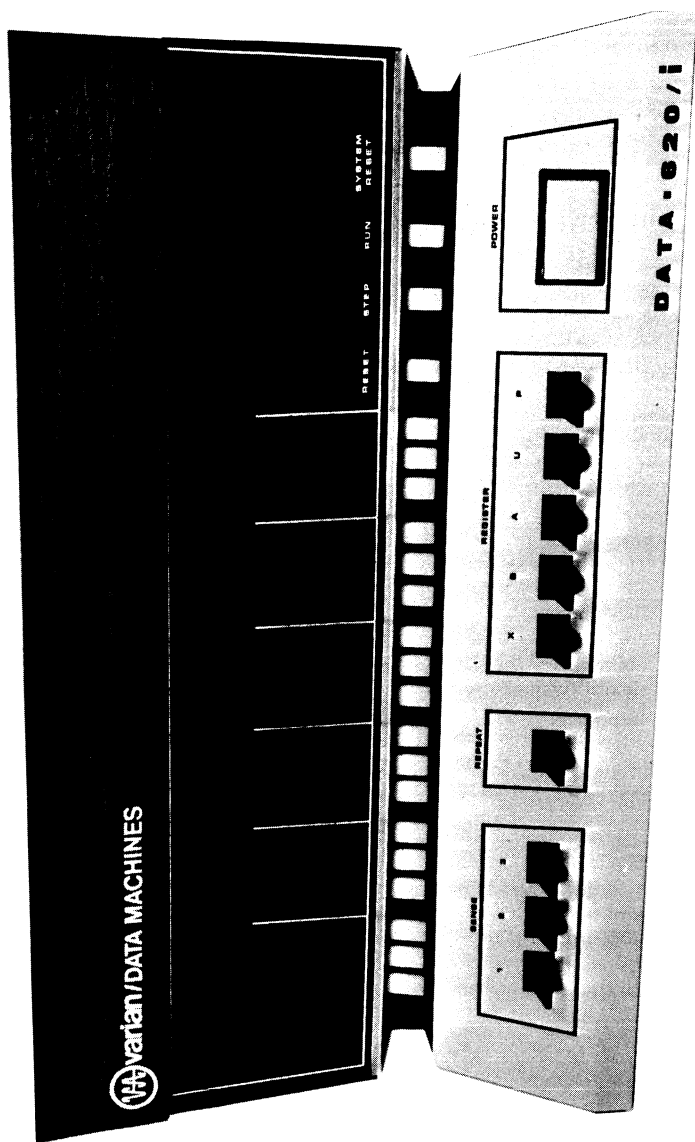


Fig. 5-1 Control Console

Table 5-1. Controls and Indicators

Control or Indicator	Function
Register Display	In-line display of 16 (or 18) bits in selected operational register. Register bits are numbered from right to left with the sign bit appearing on the far left side of the display. Lights are grouped in an octal arrangement. Selection of the register to be displayed is accomplished by the register select switches.
REGISTER Select Switches	Five alternate-action switches used to select one of five registers for display. Only one register may be selected at a time. Selection of two or more registers at the same time disables the selection logic and the display becomes blank.
Status Display	Four indicators are provided to indicate the status of the machine. OVFL indicator lights when the overflow flip-flop is set. STEP indicator lights when the computer is in the step mode and the Micro-EXEC facility is not being used. RUN indicator lights when the computer is in the run mode. ALARM indicator lights when a thermal overload condition occurs.
RESET Switch	The RESET switch causes the selected register to be cleared. This switch is disabled when the computer is in the run mode.
STEP Switch	The STEP switch is a momentary-contact switch that causes the instruction in the instruction register to be executed if the computer is in the step mode. If the computer is in the run mode, pressing the STEP switch causes the computer to halt at the completion of the instruction being executed.
RUN Switch	The RUN switch causes the program to run at the location specified by the program counter after first executing the instruction in the instruction register.
SYSTEM RESET Switch	The SYSTEM RESET switch is a system-clear control that forces the computer to the halt mode and initializes control flip-flops in the processor. In addition, all peripheral devices are initialized by SYSTEM RESET. This control is normally used as an initialize control, but is useful to halt I/O operations.

Table 5-1. Controls and Indicators (Continued)

Control or Indicator	Function
REPEAT Switch	Toggle switch that permits manual repeat of an instruction in instruction register. Pressing STEP switch executes instruction and advances program counter; however, contents of the instruction register are left unchanged. Switch on the control console is activated only when the STEP light is on (operation halted).
SENSE Switches 1, 2, 3	Toggle switches that permit manual program control whenever sense-switch-jump, jump-and-mark, or execute instructions (JSS1, JSS2, JSS3, JS1M, JS2M, JS3M, XS1, XS2, XS3) are performed. The indicated jump and execute operations are performed only if the corresponding SENSE switch is on.
POWER On/Off	Alternate-action switch/indicator that turns power supplies on and off. Indicator/switch is illuminated when power is on; indicator is off when power is off.

Instruction Repeat

In the step mode, the instruction register contains the next instruction to be executed when STEP is pressed. The program counter contains the location of the next instruction to be transferred to the instruction register after the current instruction is executed.

In some cases, it is desirable to manually execute an instruction several times. When REPEAT switch is on, instruction register loading (when STEP is pressed) is inhibited even though the instruction counter is advanced each time. This mode is particularly useful for loading words into sequential memory locations, or for displaying the contents of sequential memory locations. To load a group of sequential memory cells, set the

appropriate store-type instruction (STA, STB, STX) in the instruction register with the relative address mode in the M field and the base address in the A field. Repeated operation of the STEP switch will store the contents of the A, B or X register into sequential memory locations. The word loaded on each step may be changed by entering the desired value into the operational register for each step.

To display the contents of a group of sequential memory cells, set the appropriate load-type instruction (LDA, LDB, LDX) in the instruction register, in the relative address mode, with the base address in the P register and the A field of the U register = 0. The contents of the sequential locations will be displayed in the selected operational register with each operation of the STEP switch.

Sense Switches

The SENSE switches allow the operator to dynamically alter a program sequence in either the run or step mode. The three SENSE

switches provide a logical-AND function with bits 6-8 of the jump, jump-and-mark or execute instruction word, and consequently can be used for various logical branches selected at the console.

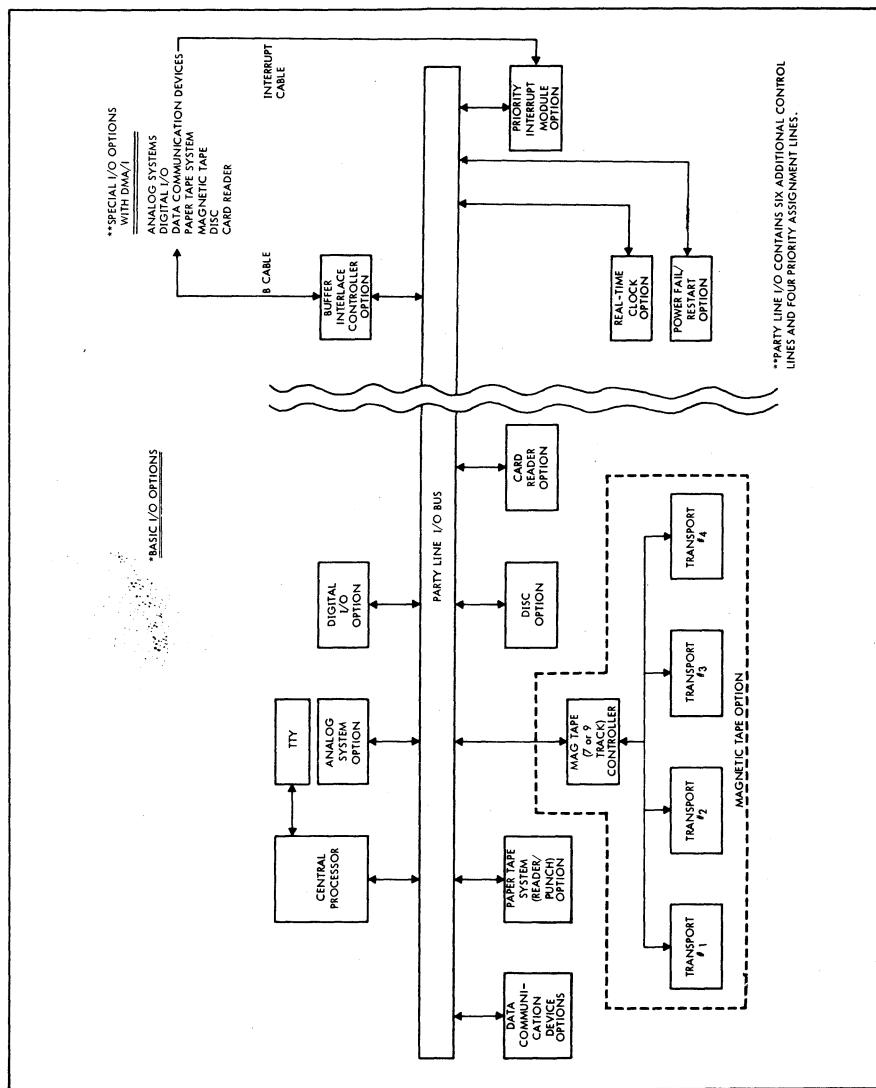


Figure 6-1. DATA 620/i Input/Output System

SECTION 6

DATA 620/i INPUT/ OUTPUT SYSTEM

6.1 GENERAL DESCRIPTION

The standard computer, without options, communications with peripheral equipment on a word-parallel (either 16- or 18-bit) input/output (I/O) bus as shown in figure 6-1. Each information transfer occurs under the control of a stored program. Information exchanges with peripheral devices are synchronized by peripheral controllers. A controller may control one or more similar peripheral devices. Each controller and the device(s) it controls comprise a peripheral device option.

Basic Operations

The basic machine provides four types of I/O operations:

External Control	An external control code is transmitted under program control from the computer to a peripheral controller.
------------------	---

Program Sense	The status of a selected peripheral controller sense line is interrogated by the computer under program control.
---------------	--

Single-Word Input Transfer	A single word of data is transferred under program control from a peripheral controller to the A register, B register or any location in memory.
----------------------------	--

Single-Word Output Transfer

A single word of data is transferred under program control to a peripheral controller from the A register, B register or any location in memory.

Direct-Memory-Access-and-Interrupt Option

A computer equipped with the direct-memory-access (DMA/I) option is capable of I/O data transfers to or from memory without program intervention. DMA/I allows external devices on the I/O bus to request data transfers on a priority basis while temporarily interrupting processing of the stored program. At the end of the current instruction the program is caused to remain idle for 3.5 microseconds during which the transfer takes place. This "cycle stealing" action does not disturb the operational registers (A, B, X, P), thus allowing the program to proceed normally to the next instruction at the conclusion of the data transfer.

The interrupt feature of the DMA/I option permits an external device to request execution of an instruction that is independent of the main program. When two or more requests are received simultaneously, the hard-wired external device priority scheme determines which device has priority. The computer is directed to a memory location specified by the interrupting device and is caused to execute the instruction found at that location. Any DATA 620/i instruction other than an I/O command may be executed in this manner.

Normally, however, the instruction will be a jump-and-mark instruction which inhibits lower priority interrupts and will call for the processing of an I/O subroutine. The lower priority interrupts may then be reenabled by an external control command.

6.2 BASIC I/O BUS SIGNAL INTERFACE

A computer equipped with the standard I/O interface can communicate directly with all peripheral device options under program control. The computer may initiate operation of a peripheral device by transmitting an external control code and a proper device address to the selected controller via the I/O bus. The computer may determine when a device is ready to send or receive information by interrogating its associated sense line. A device may be requested to place a word of data on the I/O bus during a computer input transfer or to accept a word of data placed on the bus by the computer during an output transfer.

The standard I/O bus without options consists of the E bus and five I/O control lines: FRYX-I, DRYX-I, SERX-I, IUAX-I and SYRT-I.

E Bus (EB00-I through EB17-I)

The E bus is a 16- or 18-bit, parallel, bi-directional I/O channel. It is used to transmit control codes, device addresses and data from the computer to the peripheral device options. In turn, the bus is used by these options to transmit data to the computer. A total of ten drivers and ten receivers may be connected to each line. An E-bus signal is logically true when it is at 0 vdc and is logically false when it is at +3 vdc. A typical E-bus configuration is shown in figure 6-2.

FRYX-I

Signal FRYX-I is a pulse generated by the computer to indicate that the computer has placed a device address and a control code on the E bus. Each peripheral controller examines the device address and, upon the true-to-false transition of FRYX-I, the addressed device responds to the control code. FRYX-I is logically true at 0 vdc and is logically false at +3 vdc. A total of ten receivers may be connected to the line. A typical FRYX-I line configuration is shown in figure 6-3.

DRYX-I

Signal DRYX-I is a pulse generated by the computer. During an output data transfer, DRYX-I indicates that the computer has placed data on the E bus and that the peripheral device addressed previously should strobe the data into its input buffer. During an input data transfer, DRYX-I indicates that the computer has accepted the data placed on the E bus by the peripheral device and that, following the true-to-false transition of DRYX-I, the device should remove the data. DRYX-I is logically true at 0 vdc and is logically false at +3 vdc. A total of ten receivers may be connected to the line. A typical DRYX-I line configuration is shown in figure 6-3.

SERX-I

During the execution of a program sense (SEN) command, the computer places a function code and a device address on the E bus. The addressed controller is instructed to indicate the status of the specific device condition that is identified by the function code. If the specified condition is true, the controller responds by setting the SERX-I line true. If the condition is false, SERX-I is left false. SERX-I is logically true at 0 vdc and is logically false at +3 vdc. A total of ten drivers may

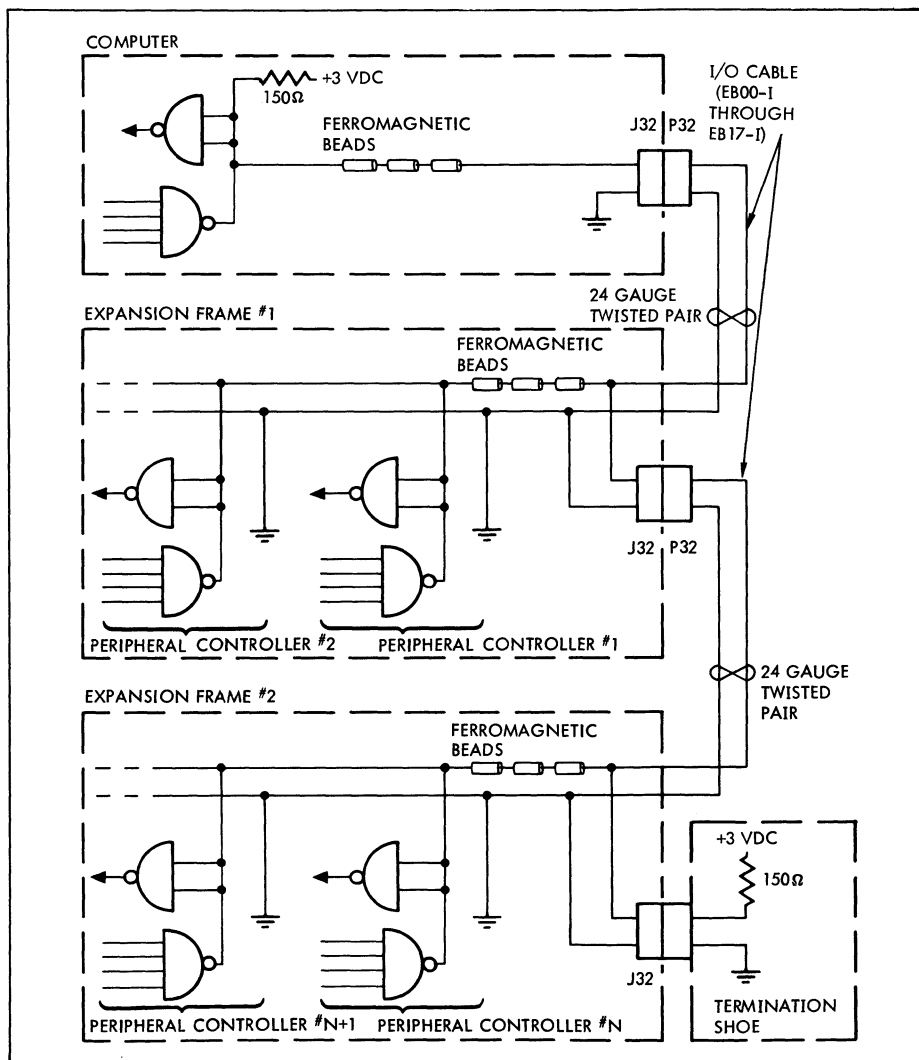


Figure 6-2. Typical E-Bus Line

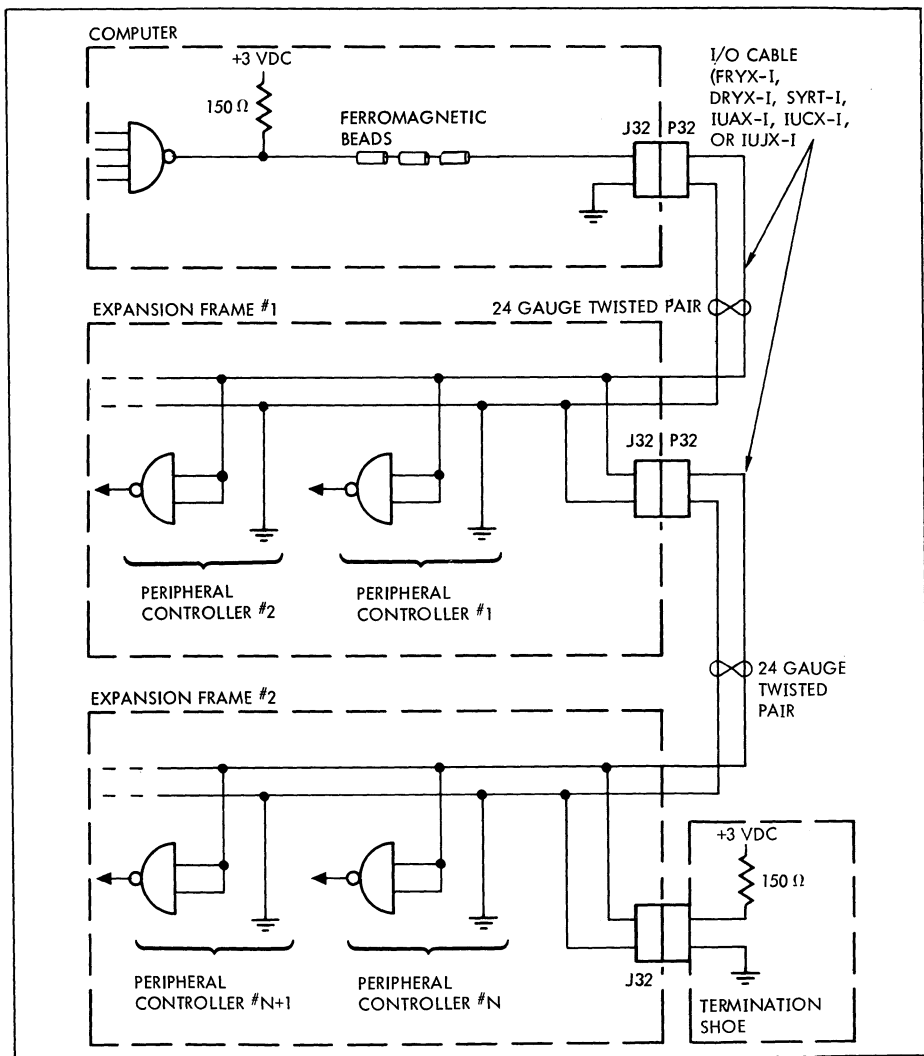


Figure 6-3. Typical Control Signal From The Computer

be connected to the line. A typical SERX-I line configuration is shown in figure 6-4.

IUAX-I

Signal IUAX-I is generated by the computer to acknowledge that either a DMA/I access or interrupt operation is in progress. Each peripheral device controller uses IUAX-I to inhibit normal device address decoding. In a basic I/O interface, without the DMA/I option, IUAX-I is held false (+3 vdc). A total of ten receivers may be connected to the line. A typical IUAX-I line configuration is shown in figure 6-3.

SYRT-I

Signal SYRT-I is used to initialize each peripheral device controller connected to the I/O bus. SYRT-I becomes true when the SYSTEM RESET switch on the control console is pressed. SYRT-I is logically true at 0 vdc and is logically false at +3 vdc. A total of ten receivers may be connected to the line. A typical SYRT-I line configuration is shown in figure 6-3.

6.3 I/O COMMAND OPERATIONS

The basic computer provides four types of I/O commands for program control of peripheral devices connected to the I/O bus. These are:

- a. External control.
- b. Program sense.
- c. Single-word input transfer.
- d. Single-word output transfer.

Table 6-1 summarizes E-bus and control signals used during these commands.

The following paragraphs describe the operations associated with each command type and the manner in which the I/O-bus signals are used. The E-bus signals used for each command execution are shown in table 6-2. The details of the instruction code formats are shown in Appendix C-10 and described in Section 4.

Device codes have been assigned to the standard DATA 620/i peripheral devices, as shown in table 6-3 and Appendix D. All device codes are in the range 00g to 77g. Each peripheral device belongs to a class, according to its function. Each class is assigned a block of codes, and specific code assignments are given to devices in that class.

External Control

The external control (EXC) command is used to initiate a specific mode of operation in a peripheral device. An example is the use of an EXC command to cause a magnetic tape transport to advance the tape one record. The format of this command is shown on page 59.

The EXC command causes the function code and device address portions of the instruction word to be placed on the E bus. The EXC I/O timing is shown in figure 6-5. Signal lines EB00-I through EB05-I indicate the device address; EB06-I through EB08-I indicate the function code. EB11-I is held true, indicating that an external control function is being performed. The device controller decodes the binary function code and, following the true-to-false transition of FRYX-I, initiates the specified mode of operation in the addressed device. During the execution of EXC, no data are exchanged between the computer and the device controller, and no response signal is expected from the controller.

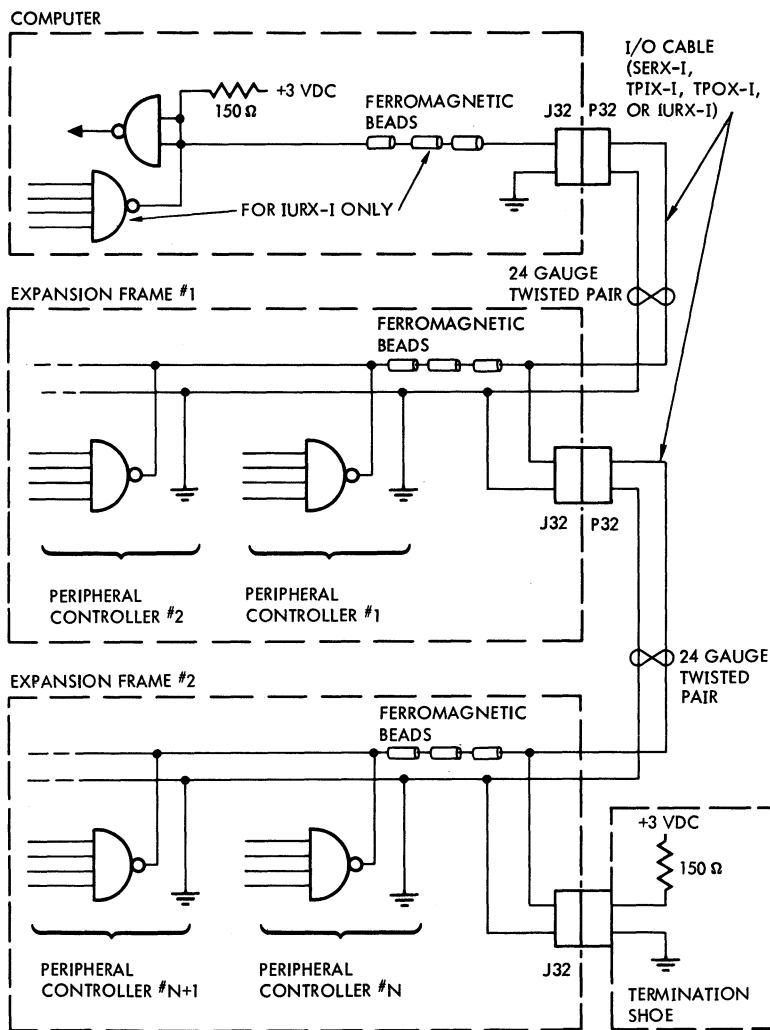


Figure 6-4. Typical Control Signal To The Computer

Table 6-2. E-Bus Signals

OPERATION	EB16-I and EB17-I	EB15-I	EB14-I	EB13-I	EB12-I	EB11-I	EB10-I, EB09-I	EB08-I	EB07-I	EB06-I	EB05-I to EB00-I
Send External Control Code to External Device		0	0	0	0	1	Unused	Function Code			Device Address (00-63 ₁₀)
Sense State of External Device		0	0	0	1	0	Unused	Function Code			Device Address (00-63 ₁₀)
Input to Memory								0	0	0	
Input to A Register		0	0	1	0	0	Unused	Note 1	0	1	Device Address (00-63 ₁₀)
Input to B Register								Note 1	1	0	
Output from Memory								0	0	0	
Output from A Register		0	1	0	0	0	Unused	Unused	0	1	Device Address (00-63 ₁₀)
Output from B Register								Unused	1	0	
Send Extended External Control Code to External Device		1	0	0	0	0	Unused	Function Code			Device Address (00-63 ₁₀)

Note 1: If EB08-I is true, selected register in computer is cleared before input.
 If EB08-I is false, selected register in computer is not cleared. The result after input is the logical -OR of the original register and the input signals.
 Bits EB06-I to EB08-I are ignored by the I/O controller during data transfers.

Table 6-3. DATA 620/i Standard Device Codes

Octal Class Codes	Octal Addresses Assigned	Peripheral or Option
00-07	00-07	ASR-33/35 Teletype Controller (serial/parallel)
10-13	10-13	Magnetic Tape Controllers
14-17	14-17	Disc Controllers
20-27	20,21	First Buffer Interlace Controller
	22,23	Second Buffer Interlace Controller
	24,25	Third Buffer Interlace Controller
	26,27	Fourth Buffer Interlace Controller
30-37	30	Card Reader
	31	Card Punch
	32	Digital Plotter
	35,36	Line Printers
	37	Paper Tape System (punch and/or reader)
40-47	40	Priority Interrupt Module
	45	Memory Protect
	47	Real-Time Clock
50-53	50-53	Optical Devices
54-57	54-57	Analog Systems
60-67	60-67	Digital Input/Output Controllers
70-77	70-77	Special Applications

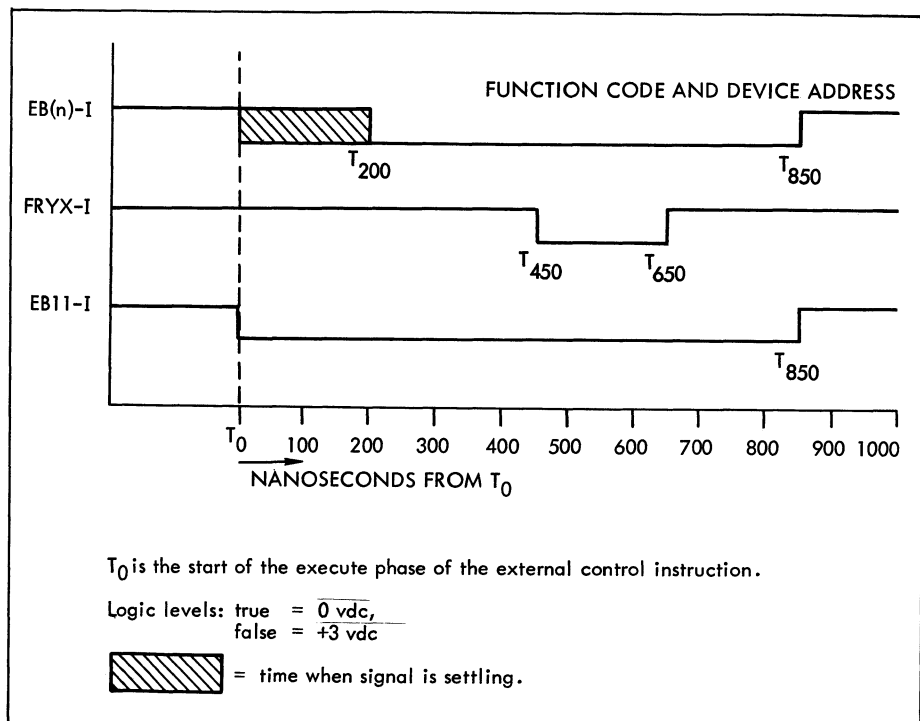


Figure 6-5. External Control Timing

When additional function codes are required, the extended EXC command (instruction code 10, 4) may be used. This command is identical to the normal EXC command (instruction code 10, 0) in timing and function, but is identified by EB15-I instead of EB11-I as shown in tables 6-1 and 6-2.

Figure 6-6 shows typical implementation of the logic required in a peripheral controller to receive, decode, and perform an EXC command.

Program Sense

The program sense (SEN) command is used to test the status of a specific device condition and, if a true condition is detected, a program jump is made. If a false condition is detected, the next instruction in sequence is executed. An example of SEN command usage is a test to determine if a magnetic tape transport is rewinding. The format of this command is shown on page 60.

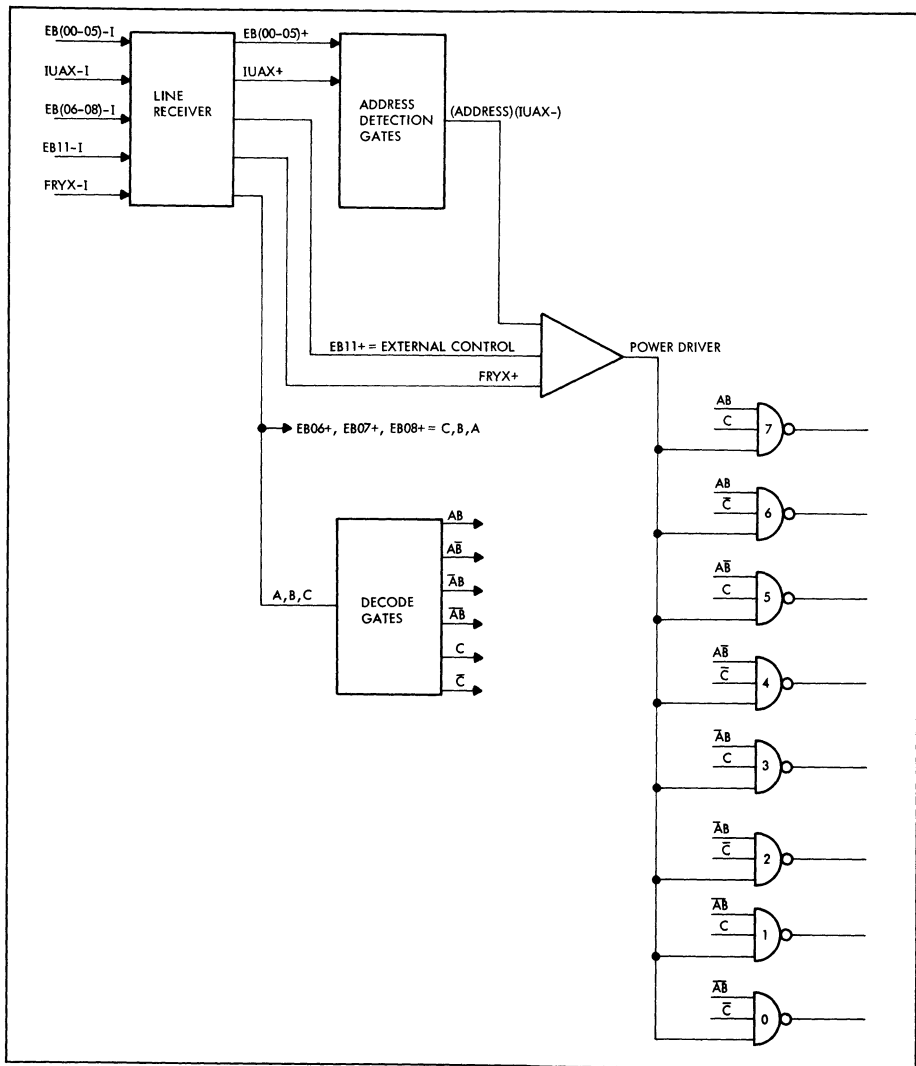


Figure 6-6. Typical Peripheral Controller Logic: EXC Command

The SEN command causes the function code and device address portions of the instruction word to be placed on the E bus. The SEN I/O timing is shown in figure 6-7. Signal lines EB00-I through EB05-I indicate the device address. EB12-I is held true to indicate that a SEN command is in progress.

Figure 5-8 shows typical implementation of the logic required in a peripheral controller to receive, decode, and respond to a SEN command. Note that the device address (usually ANDed with IUAX-I) can be used directly to enable the sense line response (SERX-I). EB12-I need not be used to enable SERX-I, since the computer samples the SERX-I line only when a SEN command is executed.

Single-Word Input Transfer

There are five instruction which provide a single word input transfer. These are:

- a. Input to A register (INA)*.
- b. Input to B register (INB)*.
- c. Input to memory (IME).
- d. Clear and input to A register (CIA).
- e. Clear and input to B register (CIB).

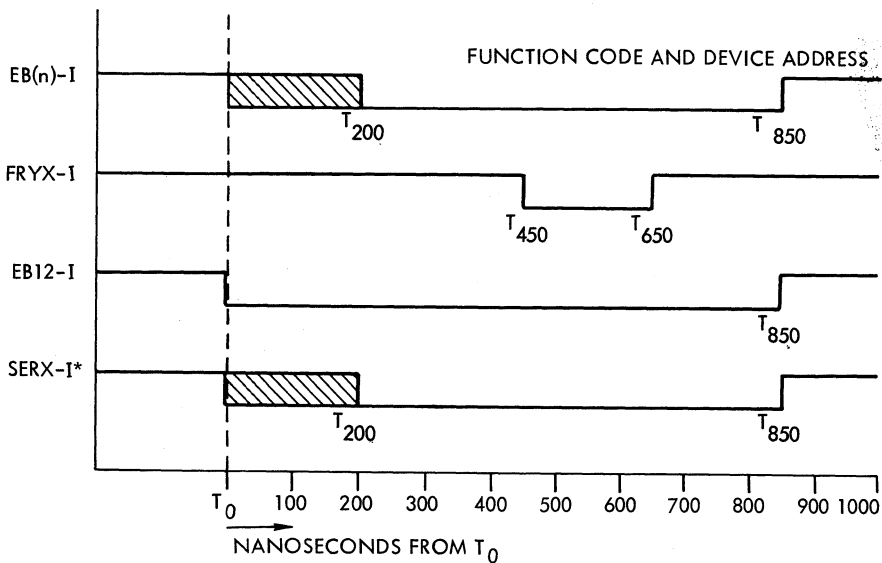
The formats of these commands are shown on pages 60 and 61. The instruction is a single word when referring to one of the registers. For input to memory, the instruction is a double word, the second of which specifies the input data word destination.

*Inclusive OR of the data lines and the specified register contents.

The execution of any one of the five single-word input transfer commands produces the same sequence of operations on the I/O bus. Consequently, in responding to the bus signals, a peripheral device controller makes no distinction between the input transfer commands.

The execution of an input transfer command is a two-phase operation. The first phase selects the peripheral device controller that will participate in the second-phase data transfer. The transfer timing is shown in figure 6-9. The first phase is initiated by the computer, which places the device address on E-bus lines EB00-I through EB05-I. EB13-I is held true during this phase to indicate that an input transfer command is in progress. Since the E bus is time-shared, a flip-flop in the peripheral controller for the selected device is set to indicate that the controller was selected and that data are to be transferred to the computer. This flip-flop, data transfer in (DTIX+), is set at the true-to-false transition (trailing edge) of FRYX-I (if the controller controls more than one device, an additional flip-flop for each device is required to identify the device selected). As the computer removes the device address and control code information, the selected controller uses DTIX+ to enable the input data onto the E bus. The controller must enable data from the selected device onto the bus no later than 850 nanoseconds after the trailing edge of FRYX-I to strobe the input data. The controller uses the trailing edge of DRYX-I to reset DTIX+, thus removing the input data from the E bus.

When the computer is transferring I/O data under program control, the transfers must be synchronized with the communicating peripheral controller. This synchronization is accomplished by sampling the state of the controller for a ready condition by issuing a sense command prior to the data transfer command.



T_0 is the start of the execute phase of the sense instruction.

Logic levels: true = 0 vdc,
false = +3 vdc.

 = time when signal is settling.

* SERX-I is normally on at T_{200} ; it must be on by T_{650} .

Figure 6-7. Sense Response Timing

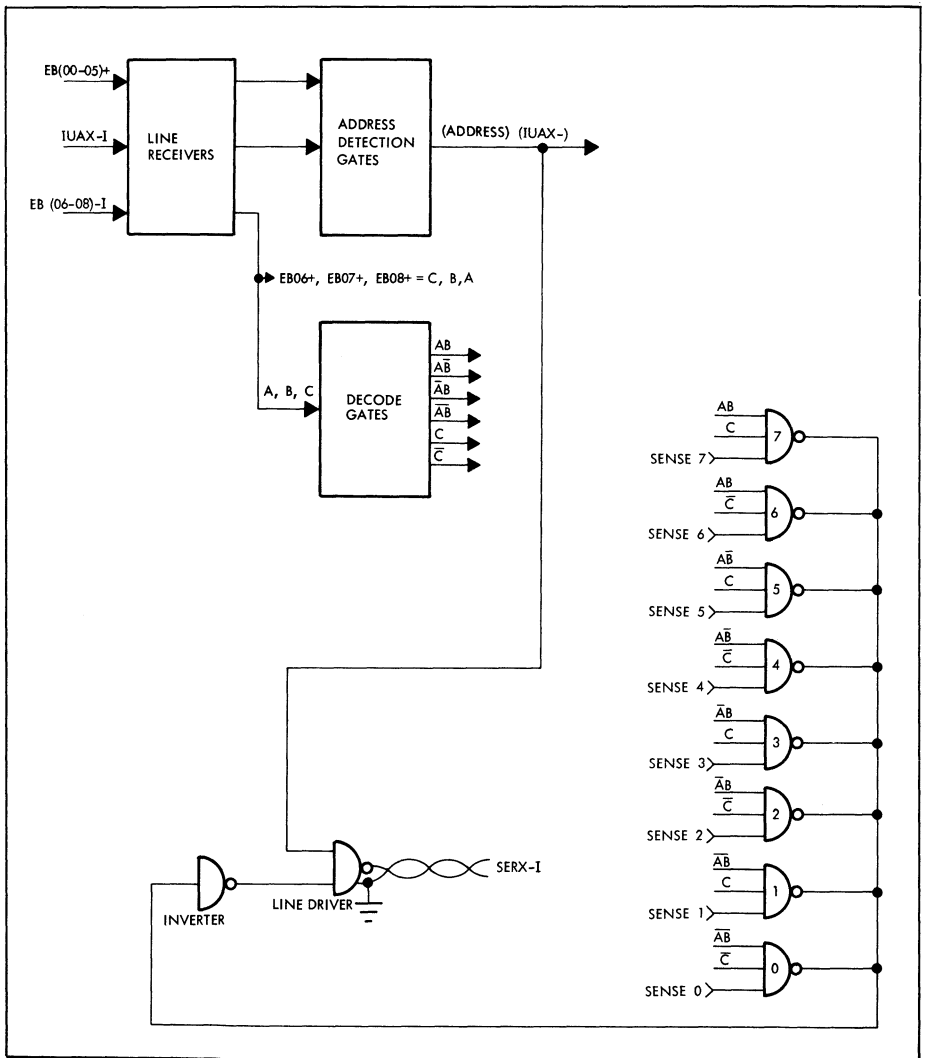
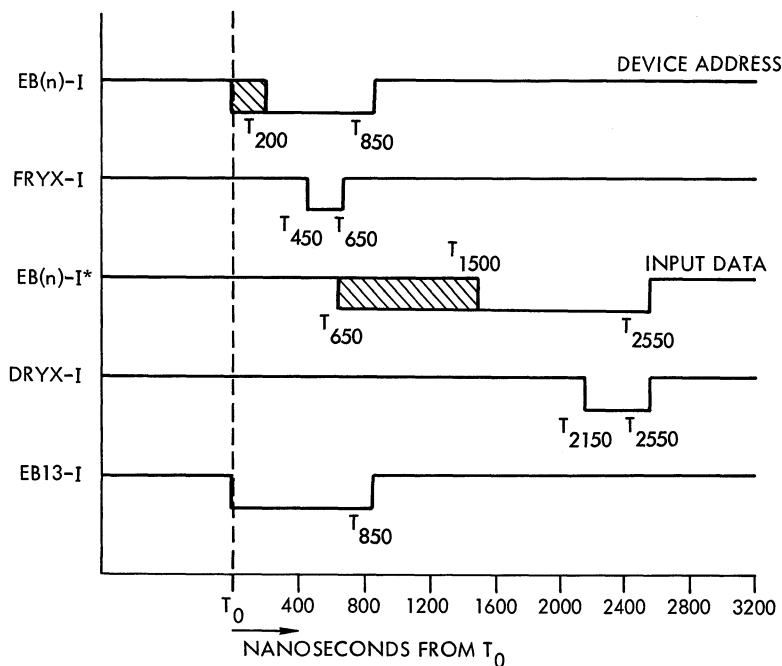


Figure 6-8. Typical Peripheral Controller Logic: SEN Command



T_0 is the start of the execute phase of the data transfer in instruction.

Logic levels: true = 0 vdc,
false = +3 vdc.

 = time when signal is settling.

* EB(n)-I (input data) must be off by T_{2700} .

Figure 6-9. Data Transfer In Timing

Figure 6-10 shows typical implementation of the logic required in a peripheral controller to perform an input data transfer.

Single-Word Output Transfer

There are three single-word output commands. These are:

- a. Output from A register (OAR).
- b. Output from B register (OBR).
- c. Output from memory (OME).

The format of these commands is shown on pages 61 and 62. The instruction is a single word when referring to one of the registers. For output from memory, the instruction is a double word, the second of which specifies the output data-word source location.

The sequence of signals placed on the I/O bus is the same for each of the three output commands, and the participating peripheral device controller makes no distinction between the output commands.

The execution of an output command is a two-phase operation similar to that described for an input command. The first phase selects the

peripheral device which will participate in the second-phase data transfer. The transfer timing is shown in figure 6-11. The first phase is initiated by the computer, which places the device address on E-bus lines EB00-I through EB05-I. EB14-I is held true during this phase to indicate that an output transfer command is being executed. A flip-flop, data transfer out (DTOX+), in the controller for the selected device is set at the trailing edge of FRYX-I (if the controller controls more than one device, an additional flip-flop for each device is required to identify the device selected). Following FRYX-I, the computer removes the device address and control code, and places the output data on the E-bus lines. DTOX+ is used by the controller to gate the contents of the E bus into an input buffer at the trailing edge of DRYX-I. The trailing edge of DRYX-I is also used to reset DTOX+.

When the computer is transferring I/O data under program control, the transfers must be synchronized with the communicating peripheral controller. This synchronization is accomplished by sampling the state of the controller for a ready condition by issuing a sense command prior to the data transfer command.

Figure 6-12 shows typical implementation of the logic required in a peripheral controller to perform an output data transfer.

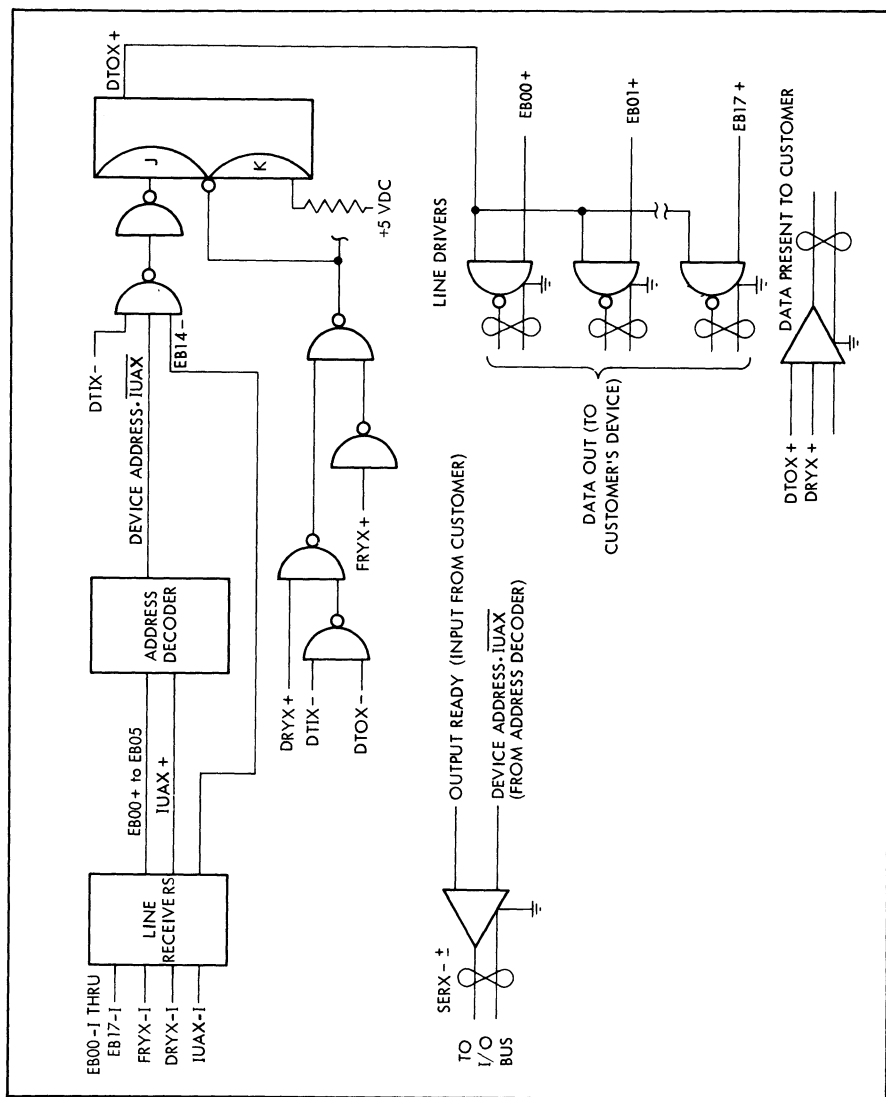
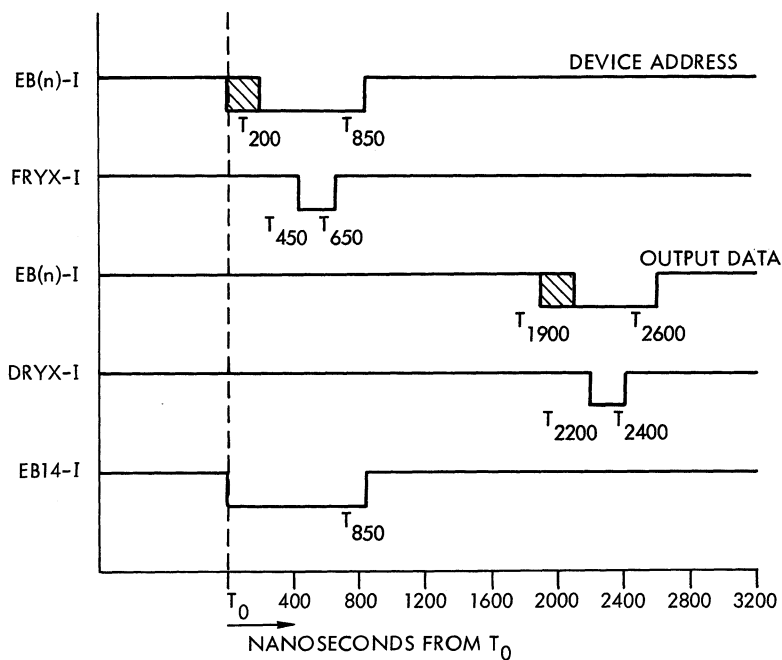


Figure 6-10. Typical Peripheral Controller Logic: Input Data Transfer



T_0 is the start of the execute phase of the data transfer out instruction.

Logic levels: true = 0 vdc,
false = +3 vdc.

 = time when signal is settling.

Figure 6-11. Data Transfer Out Timing

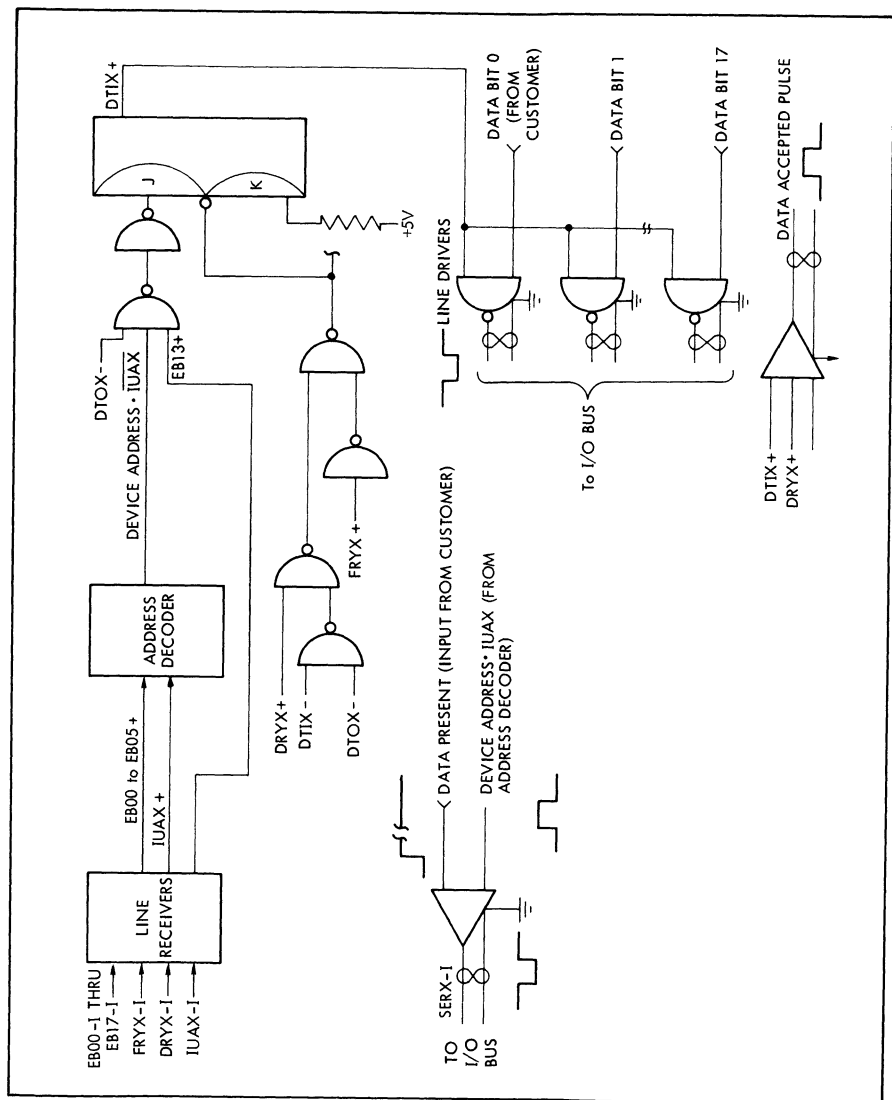


Figure 6-12. Typical Peripheral Controller Logic: Output Data Transfer

SECTION 7

DIRECT-MEMORY-ACCESS-AND-INTERRUPT OPTION

GENERAL DESCRIPTION

The direct - memory - access - and - interrupt (DMA/I) option of the DATA 620/i provides the control necessary to perform automatic (non-programmed) data transfers to and from memory, and to perform automatic interrupts. This option is required for use with the buffer-interlace - controller option, the power fail/restart option and the priority interrupt option. These options are described in Section 9. The DMA/I may also be used to control automatic data transfers and interrupts when using specially designed peripheral controllers.

7.1 DIRECT-MEMORY-ACCESS FUNCTION

The DMA function provides the capability of data transfers to and from memory without program intervention. It allows a peripheral device option to request the transfer of one word of data while temporarily halting the processing of the stored program. This halt while transferring data is referred to as "trapping". To prevent several peripheral device options from requesting data transfers at the same time, a hard-wired priority scheme determines the order in which the requests may occur. During the interval in which the transfer takes place, the stored program is stopped for 3.5 microseconds (see Section 8 for detailed timing of the transfer). This cycle stealing does not disturb the contents of the operational registers (A, B, X, P). In this way, the program may proceed normally at the conclusion of the transfer. With the DMA feature and a controller designed to properly interface with

it (such as a buffer interlace controller), the data-transfer rate can be as high as 202,000 words per second, as follows:

DMA trapping sequence:	2.7 microseconds
Synchronization time:	0.45 microseconds
Minimum instruction time:	<u>1.8 microseconds</u>
Period of transfer rate:	4.95 microseconds

Rate = $1/4.95 = 202,000$ words per second.

7.2 INTERRUPT FUNCTION

The interrupt function of the DMA/I option provides the capability for peripheral device option to request the computer to execute an instruction independent of the stored computer program. In order to prevent several peripheral device options from requesting interrupts at the same time, a hard-wired priority scheme, which is also used for the DMA function, determines the order in which the requests may occur (such requests will normally originate in priority interrupt modules). During the interrupt, the computer is directed to the memory location specified by the interrupting device and is caused to execute the instruction found at that location. The execution of any instruction other than an I/O command may be requested. Normally, the instruction will be a jump-and-mark command which inhibits

other interrupts and will call for the processing of an I/O subroutine. In this event, the interrupt system is automatically inhibited until the inhibit is terminated by an external control instruction.

7.3 DIRECT-MEMORY-ACCESS-AND- INTERRUPT CONTROL LINES

The following control lines, along with all of the standard E-bus controls, are used with the DMA/I options: IUAX-I, IURX-I, TPOX-I, TPIX-I, IUCX-I and IUJX-I. These control lines are logically true at 0 vdc and logically false at +3 vdc. A total of ten receivers or drivers may be connected to each control line.

IUAX-I (Interrupt Acknowledge)

This control signal is set true by the DATA 620/i to acknowledge the receipt of an interrupt, a trap in, or a trap out. The interrupting or trapping device controller can communicate an address to the computer and can receive data from or send data to the computer only when this control signal is true. IUAX-I is also used to inhibit device address decoding in every device controller during the address phase of an interrupt or DMA operation. This prevents the controllers from interpreting the lower-order bits of a memory address as a device address.

IURX-I (Interrupt Request)

By setting IURX-I true, the interrupting device controller requests the DATA 620/i to execute an instruction. The address of this instruction is placed on the E bus by the interrupting device upon receipt of IUAX-I from the computer.

TPOX-I (Trap-Out Request)

By setting TPOX-I true, the trapping device controller requests the DATA 620/i to output

one word of data from memory. The address of this word is placed on the E bus by the controller upon receipt of IUAX-I from the computer.

TPIX-I (Trap-In Request)

By setting TPIX-I true, the trapping device controller requests the DATA 620/i to input one word of data to memory. The address of this word is placed on the E bus by the controller upon receipt of IUAX-I from the computer.

IUCX-I (Interrupt Clock)

This control signal is a 1.1-MHz clock from the DATA 620/i that is disabled by IUAX-I. When IUAX-I is false, the clock is present on the IUCX-I line. When IUAX-I is true, IUCX-I is held true. The true-to-false transition of IUCX-I is used to set the request flip-flops (IURX-I, TPOX-I, TPIX-I) in respective controllers.

IUJX-I (Interrupt Jump)

This control signal from the computer is used to inhibit all interrupts that occur after a jump-and-mark instruction when that instruction is the result of an interrupt request. This signal becomes true 2.7 microseconds from the false-to-true transition of IUAX-I and remains true for 450 nanoseconds.

7.4 DETAILS OF DIRECT-MEMORY- ACCESS SEQUENCE

A controller using the DMA option must be able to generate or respond to the following signals: IUAX-I, TPOX-I, TPIX-I, IUCX-I, FRYX-I, DRYX-I and EBNX-I, plus priority signals on the PRMX-I input and on the PRNX-I output.

The DMA (trap in/out) sequence is comprised of three phases: request, address and data, as shown in the DMA timing sequence, figures 7-1 and 7-2.

Request Phase

The peripheral controller of the requesting device option having the highest priority generates a cycle stealing request (sets TPOX-I or TPIX-I true) on the trailing edge of the interrupt clock (IUCX-I) and waits for an interrupt acknowledge (IUAX-I) from the computer.

Address Phase

Interrupt acknowledge (IUAX-I) is received by the peripheral controller of the requesting device, thereby holding IUXC-I true. The controller then places the memory address, into which or from which data are to be read, onto the E bus and waits for the trailing edge of FRYX-I.

Data Phase

At the trailing edge of FRYX-I, the controller removes the address from the E bus. Data are then placed on the E bus by either the computer or the requesting device controller. These data are gated into the device or into the computer at the trailing edge of DRYX-I. All signals are removed from the bus when IUAX-I becomes false.

7.5 DETAILS OF INTERRUPT

A controller communicating with the interrupt portion of the DMA-I option must be able to generate or respond to the following signals: IUAX-I, IUCX-I and EBNX-I, plus priority signals on the PRMX-I input and on the

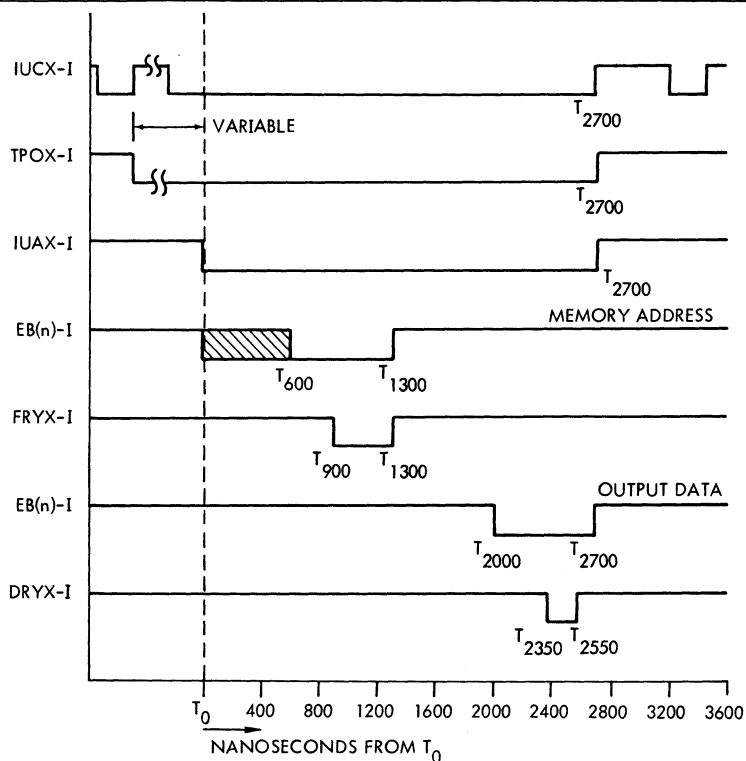
PRNX-I output. The interrupt sequence is comprised of two phases: request and address, as shown in the interrupt timing sequence, figure 7-3.

Request Phase

The peripheral controller of the requesting device options having the highest priority generates an interrupt request (sets IURX-I true) on the trailing edge of the interrupt clock (IUCX-I). The controller then waits for an interrupt acknowledge (IUAX-I) from the computer.

Address Phase

When the DATA 620/i has recognized the request, it responds by setting IUAX-I true, thus holding IUXC-I true. Upon receipt of IUAX-I from the computer, the requesting device controller places on the E bus the address containing the instruction to be executed. The instruction at this address may be any instruction except an I/O instruction. In the case of a two-word instruction, the DATA 620/i ORs a binary one into bit position zero of the second word, making this address an odd number. Because of this, the address placed on the E bus by the requesting device controller must be an even number. If the instruction is a jump-and-mark instruction, the controller receives an interrupt-jump signal (IUJX-I) from the computer. IUJX-I resets the master enable in the controller of every device capable of making a request, thereby inhibiting further interrupts from these controllers during the subsequent subroutine. Each controller must be re-enabled by the computer program at the end of the subroutine. The address phase ends when IUAX-I becomes false. All signals must be removed from the bus at this time.



T_0 is the start of the output sequence.

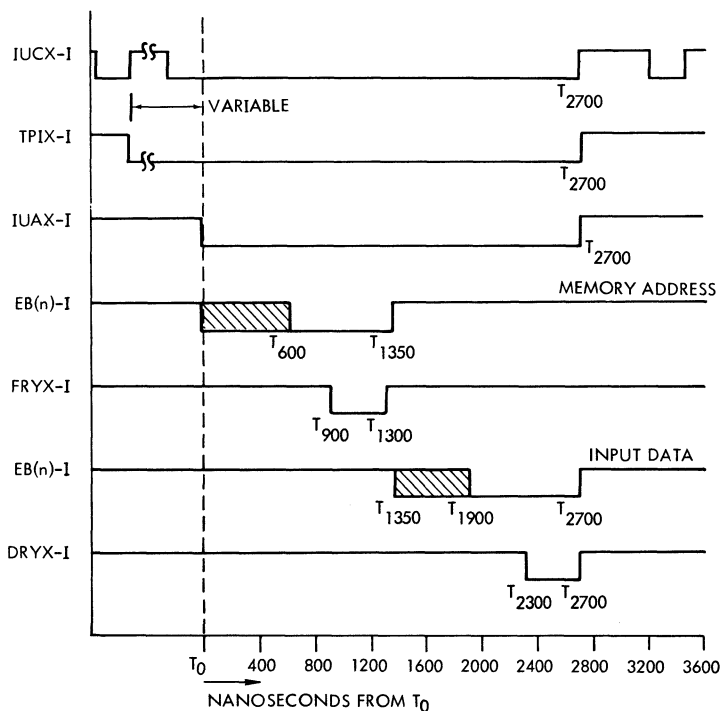
Logic levels: true = 0 vdc,
false = +3 vdc.

 = time when signal is settling.

TPOX-I is normally off at T_{2700} ; it must be off by T_{2900} .

EB(n)-I (memory address) is normally on at T_0 ; it must be on by T_{600} . It is normally off at T_{1300} ; it must be off by T_{1500} .

Figure 7-1. Output Timing



T_0 is the start of the input sequence.

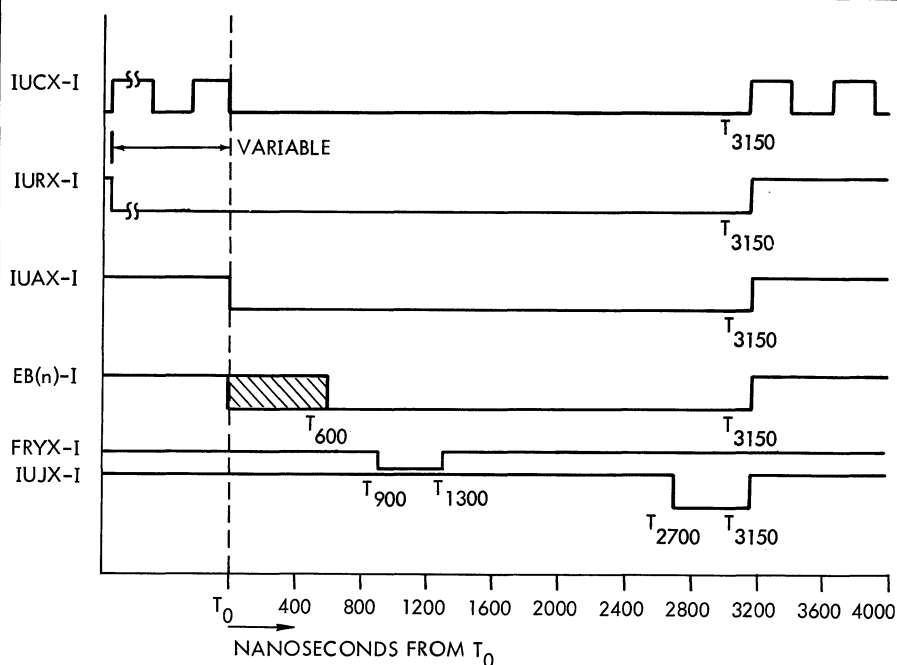
Logic levels: true = 0 vdc,
false = +3 vdc.

TPIX-I is normally off at T_{2700} ; it must be off by T_{2900} .

EB(n)-I (memory address) is normally on at T_0 ; it must be on by T_{600} . It is normally off at T_{1300} ; it must be off by T_{1600} .

EB(n)-I (input data) is normally on at T_{1350} ; it must be on by T_{1900} . It is normally off at T_{2700} ; it must be off by T_{2900} .

Figure 7-2. Input Timing



T_0 is the start of the timing and interrupt sequence.

Logic levels: true = 0 vdc,
false = +3 vdc.

 = time when signal is settling.

IURX-I is normally off at T_{3150} ; it must be off by T_{3300} .

EB(n)-I is normally on at T_0 ; it must be on by T_{600} . It is normally off at T_{3150} ; it must be off by T_{3300} .

IUJX-I is present for a jump and mark instruction.

Figure 7-3. Interrupt Timing

7.6 EXTERNAL DEVICE PRIORITY

Every peripheral device option on the I/O bus which is capable of automatic requests (interrupt, trap in and trap out) must be assigned a level of priority. This is to assure that only one such option at a time will be initiating an automatic request. Only options capable of initiating automatic requests require a priority assignment.

The priorities assigned determine the electrical location of each peripheral device option on the I/O bus. Priority lines PR1X-I, PR2X-I, PR3X-I and PR4X-I in the various I/O cables are used to place the options in the assigned order on the I/O bus. This is graphically shown in figures 7-4 and 7-5. The simplest interconnection results when the physical position corresponds directly to the priority assignment as in figure 7-4. In this case, the PR2X-I and PR3X-I lines are not used. When the physical location on the cable does not correspond directly to the priority assignment, a more complex interconnection results, such as in figure 7-5. The controller of each option contains priority logic to determine when that option has priority. Typical priority logic is shown in figure 7-6.

In figure 7-6, IURX-I, TPIX-I and TPOX-I are the interrupt, trap-in and trap-out

request signals, respectively. These signals have previously been described.

PRMX-I

PRMX-I is the priority-in input to the controller of each peripheral device option. When this signal is true (0 vdc) for a given option, no option having a higher priority has requested an interrupt or trap. The controller of the given option can then initiate a request if its request flip-flop is set. When PRMX-I is false (+3 vdc) for a given option, an option having a higher priority is performing a request. The controller of the given option is therefore inhibited from initiating a request, regardless of the state of its request flip-flop. If an option having a higher priority initiates a request before a request from a lower-priority option is acknowledged (i.e., before IUAX-I becomes true), the higher-priority option assumes control.

PRNX-I

PRNX-I is the priority-out output from the controller of each peripheral device option. This signal is false (+3 vdc) from the controller of a given option if either the given option or one with a higher priority is initiating a request. When false, this signal prevents all lower-priority options from initiating requests.

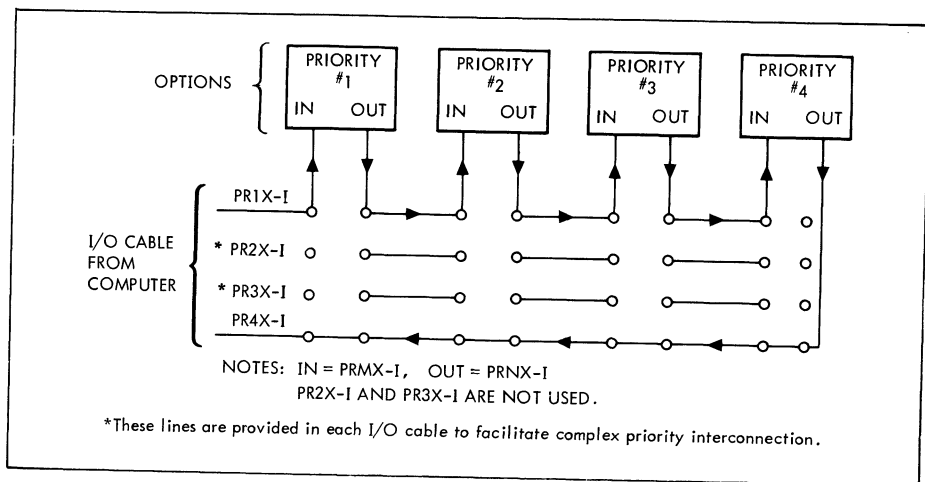


Figure 7-4. Device Interconnection: Device Position Corresponds to Priority Assignment

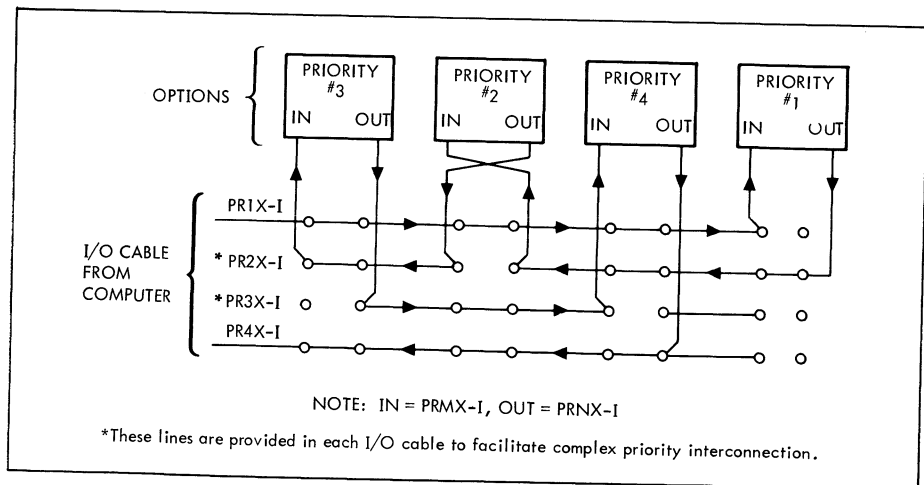


Figure 7-5. Device Interconnection: Device Position Does Not Correspond to Priority Assignment

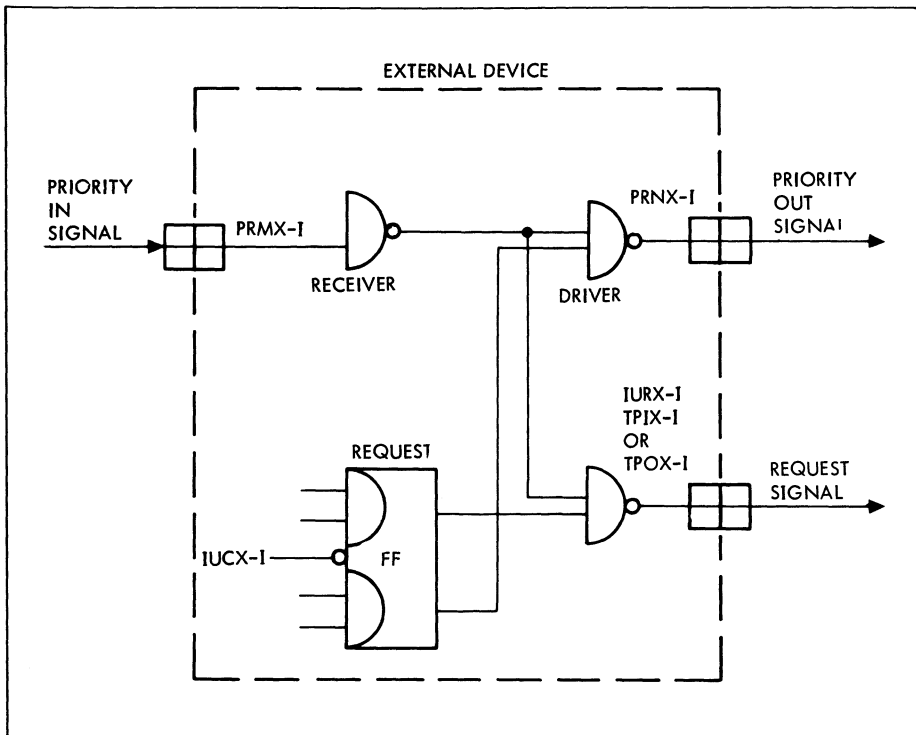


Figure 7-6. Typical Priority Logic of an External Device

SECTION 8

I/O CABLE INTERFACE CHARACTERISTICS

8.1 I/O BUS LINE CONFIGURATIONS

The I/O-bus interface generally involves three types of driver/receiver configurations. The first configuration is represented by the bidirectional E bus. The time-shared nature of the E bus requires that the computer and most peripheral controllers employ one cable driver and one cable receiver for each line of the bus. A typical E-bus configuration is shown in figure 6-2. E-bus communication is such that information is transferred from the computer cable driver to a peripheral controller cable receiver or from a controller driver to the computer receiver. Communication between controllers does not occur on the E bus. Each bus line is terminated by 150 ohms to +3 vdc at the computer, and in a termination shoe at the opposite end of the I/O cable. Ferromagnetic beads (VDM P/N 15A0005) are used to control the rise and fall times of the E-bus signals. It is necessary to place three beads in the vicinity of the signal driver to insure satisfactory signal control; consequently, beads are placed on the driver outputs as shown in figures 6-2, 6-3, and 6-4.

8.2 I/O CABLE DRIVER AND RECEIVER

The I/O cable driver is a four-input NAND gate of the Texas Instruments SN15-844N series (or equivalent) which has been selected so that the saturated common-emitter output stage is capable of conducting up to 70 milliamperes without exceeding a saturation voltage of 0.5 vdc. This driver can be

collector-tied with another driver to produce a negative-logic OR function. The driver circuit schematic is shown in figure 8-1.

The I/O cable receiver is a Texas Instruments series SN15-846N two-input NAND gate (or equivalent). The receiver circuit schematic is shown in figure 8-2. The receiver input represents a maximum load (current source) of 1.4 milliamperes at 0 vdc. The receiver input switching threshold is 1.5 ± 0.5 vdc.

8.3 I/O CABLE AND CONNECTOR

The I/O cable contains 37 twisted-pair, 24-gauge stranded wires. The I/O cable is composed of several cable segments. The first segment extends from the plug, P32, on the computer to receptacle J32 on the first expansion frame (or the customer chassis). This is shown in figure 6-2, 6-3, or 6-4. The second segment extends from the cable plug, P32, on the first expansion frame to the receptacle, J32, on the second expansion frame. The termination shoe is shown in figure 6-2, 6-3, or 6-4 as being mounted on receptacle J32 on the second expansion frame. When a third expansion frame is added, the termination shoe must be removed, the interconnecting cable attached to J32, and the termination shoe mounted on receptacle J32 of the third expansion frame. Similarly, when special external equipment is connected to the I/O cable, the termination shoe is mounted on the special equipment. The I/O cable connector pin assignments are listed in table 8-1. Total cable length is limited to 20 feet.

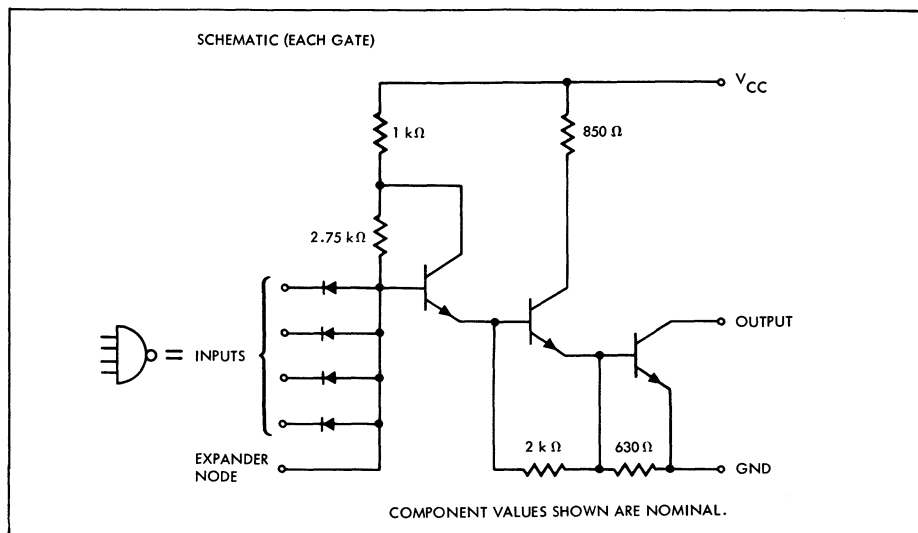


Figure 8-1. I/O Cable Driver

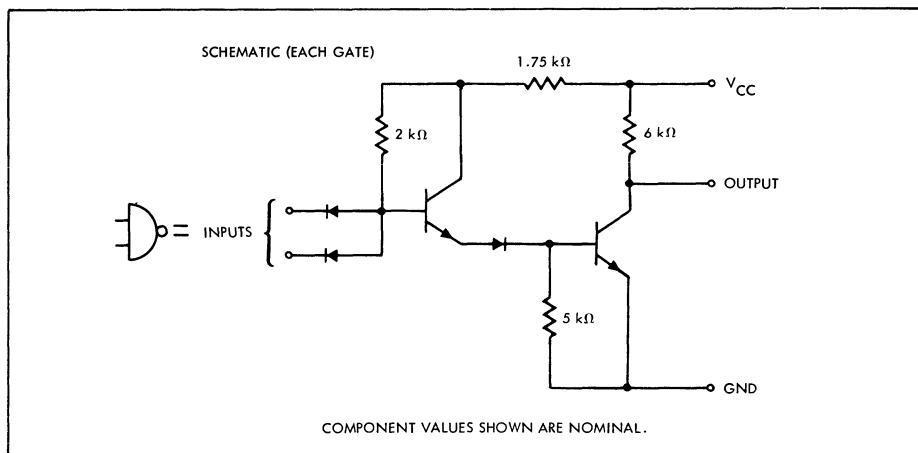


Figure 8-2. I/O Cable Receiver

Table 8-1. I/O Cable Connector Pin Assignments

Line	Pin	Function
1	1	EB00-I
2	2	R
3	3	EB01-I
4	4	R
5	5	EB02-I
6	7	R
7	8	EB03-I
8	10	R
9	11	EB04-I
10	12	R
11	13	EB05-I
12	14	R
13	15	EB06-I
14	16	R
15	17	EB07-I
16	18	R
17	20	EB08-I
18	21	R
19	22	EB09-I
20	23	R
21	24	EB10-I
22	25	R
23	26	EB11-I
24	27	R
25	28	EB12-I

Line	Pin	Function
26	29	R
27	30	EB13-I
28	31	R
29	32	EB14-I
30	33	R
31	34	EB15-I
32	35	R
33	36	EB16-I
34	37	R
35	38	EB17-I
36	39	R
37	40	FRYX-I
38	41	R
39	42	DRYX-I
40	43	R
41	44	SERX-I
42	45	R
43	46	TPIX-I
44	47	R
45	48	TPOX-I
46	49	R
47	50	PR1X-I
48	51	R
49	52	PR2X-I
50	53	R

Line	Pin	Function
51	54	PR3X-I
52	55	R
53	56	PR4X-I
54	57	R
55	58	SYRT-I
56	59	R
57	60	IUAX-I
58	62	R
59	63	IUCX-I
60	64	R
61	65	IURX-I
62	66	R
63	67	IUJX-I
64	70	R
65	71	
66	72	
67	73	
68	74	
69	75	
70	76	
71	77	
72	78	
73	79	
74	80	+3 vdc
75	82	GND

SECTION 9 OPTIONS

This is a brief discussion of the various options that can be added to the basic DATA 620/i. More complete discussion of these options will be found in separate manuals available from Varion Data Machines.

The following options can be mounted in the basic computer rack.

9.1 MULTIPLY, DIVIDE, EXTENDED ADDRESSING

This option provides three additional features for the computer. These are: multiply, divide and extended addressing.

During multiply, the contents of the B register are multiplied by the contents of a specified memory location. The original contents of the A register are added to the final product. Execution time is 18 microseconds for the basic 16-bit computer; 19.8 microseconds for the 18-bit model.

During divide, the contents of the A and B registers are divided by the contents of a specified memory location. Execution time is 18 to 25.2 microseconds for the basic 16-bit computer; 19.8 to 28.8 microseconds for the 18-bit model.

During extended addressing, all single-word instructions can be programmed as double-word instructions, where the second word contains the effective address of the operand. This option is used with the basic DATA 620/i.

9.2 MEMORY/PERIPHERAL CONTROLLER EXPANSION CHASSIS

This option provides the necessary power supply and mounting hardware required for an additional memory module and/or a peripheral controller chassis. The chassis (backpanel wiring) is divided into halves. Each half can accommodate a 4096-word memory module. Alternately, a peripheral control chassis may be installed in the right half. Each peripheral controller chassis can contain up to four controllers.

This option requires 10-1/2 inches of height in a 19-inch rack and mounts below the mainframe.

9.3 MEMORY MODULE

This option is a 4096-word (16-bit) memory module that provides additional on-line core storage for the standard DATA 620/i computer. The memory has a cycle time of 1.8 microseconds and utilizes a unique stack-temperature compensation scheme that does not require a stack heater.

This concept allows stack temperature to follow ambient temperature but compensate by controlling drive circuits with a simple and unique electronic servo. This servo senses stack temperature and automatically adjusts drive and inhibit currents to their optimum values. This method avoids operation near marginal limits and makes the memory instantly available regardless of ambient temperature.

The memory is expandable to 32,768 words in 4096-word increments. This memory option requires one or more expansion chassis (9.2 above). Two memory modules can be contained in an expansion chassis. Up to seven memory modules can be on-line to the computer.

9.4 DIRECT MEMORY ACCESS AND INTERRUPT

This option DMA/I provides "cycle-stealing" capability to the party-line I/O system. It permits external devices to request service from the computer on a priority basis and to interrupt the computer for 2.7 microseconds while the memory is cycled. DMA/I permits data transfers to occur at a rate of over 200,000 words (16 or 18 bits) per second. This operation does not disturb the operational registers (A, B, X, P) of the computer, thus allowing the program to proceed normally at the conclusion of the data transfer. This option is physically mounted in the DATA 620/i mainframe.

9.5 REAL-TIME CLOCK

The real-time clock provides a flexible time-orientation system that can be used in a variety of real-time functions, including time-of-day accumulation and as an interval timer.

The real-time clock can generate two interrupts. The first interrupt is a time-base signal that increments a specific memory location when recognized by the computer. The second interrupt occurs when the incremented memory location reaches a count of 40,000. The frequency of the first interrupt can range from 100 Hz to 10 kHz, or an external frequency source can be used. This option is physically mounted in the DATA 620/i mainframe. Direct memory access and interrupt must be installed before this option may be used.

9.6 POWER FAIL/RESTART

This option permits automatic recovery and restart of a program when ac line power to the computer is discontinuous.

A power failure is detected when the 115-vac supply falls below an adjustable threshold (105 vac). Any time a power failure is detected, a power-fail interrupt is generated, and memory-data-save and processor-reset operations are initiated before dc power falls below operating level.

This option is installed in the DATA 620/i mainframe.

9.7 PRIORITY INTERRUPT

This option provides the DATA 620/i with a multi-level priority interrupt system that includes single-instruction execute, group enable/disable and selective arm/disarm capability. Each interrupt line is assigned a unique memory destination address which is the first of a pair of locations. The system is modular and expendable in groups of eight levels. This option is mounted in the DATA 620/i mainframe or in an expansion chassis. (Option 9.2)

The interrupt system is automatically scanned every 900 nanoseconds and the interrupt is recognized before the fetch cycle of the next instruction to be executed. If signals exist on one or more interrupt line, the highest-priority interrupt is recognized.

Each group of eight interrupt can be enabled/disabled individually and contains an eight-bit mask register that controls the individual interrupt lines. Acknowledgment of an interrupt by the computer causes the instruction-specified memory address of the interrupt to be accessed. The instruction can be any of the DATA 620/i repertoire. This technique permits an interrupt to be serviced in one instruction period. If the executed instruction

is jump and mark, the interrupt system is automatically inhibited, permitting the inhibit to be terminated under program control.

The DATA 620/i interrupt system provides high-speed reaction time, expansion capability and arm/disarm versatility for real-time control.

9.8 MEMORY PROTECT SYSTEM MODULE

The Memory Protect System consists of a 16-bit mask register, which identifies either:

- a. 16 contiguous sectors of 256 words (4K system).
- b. 16 contiguous sectors of 512 words (8K system).
- c. 16 contiguous sectors of 1024 words (16K system).
- d. 16 contiguous sectors of 2048 words (32K system).

The high order four bits of every address are decoded into 16 discrete lines. These lines are brought in a one-to-one correspondence with the mask register.

The sequence of events which occurs when the memory protect system is enabled is:

1. The address is placed on the L-bus (memory location).
2. The address is automatically decoded into 1 of 16 possible segments. This is compared with the corresponding bit in the memory protect mask.
3. Should this bit of the protect mask be in the protect state, then only a read/restore cycle will be permitted. If the cycle request was for a read, the request will be executed and no violation signal will be generated. If the cycle

request was for a store, a violation signal will be generated and a read cycle executed.

This option permits a top priority executive, control, alarm, processing, or monitor system to remain resident in memory while other programs are being processed on a time-shared basis.

9.9 NEGATIVE I/O

The negative I/O option module consists of a bidirectional level shifter for each signal on the E bus I/O channel. An E bus signal is logically true when it is at 0 vdc and is logically false when it is at +3 vdc. Positive logic from the DATA 620/i is converted negative logic or negative logic from the user is converted to positive logic for transmission on the I/O line at levels of 0 and -6 vdc.

The 140 pin cable connector available with this module is in addition to the standard I/O cable, thereby allowing the use of both positive and negative interfaces with peripheral equipment. In particular, various external equipments available for the DATA 620 and DATA 620A can also be connected to the DATA 620/i if the Negative I/O module is used.

9.10 MICRO-EXEC

Micro-EXEC is the technique of using the computer's micro-functions (set A-Register, Start Memory Cycle, output on I/O bus, etc.) to form complex macro-functions. Control of memory, registers, bus connections, input/output, adder and shift logic is available externally to the user through Micro-EXEC.

In any stored program computer, a large amount of time is spent accessing the stored

program instructions. Special purpose computers using routines designed for the particular task eliminate the instruction accessing by "hardwiring" the instructions in the computer logic. Generally, special purpose computers are fast, efficient, expensive and inflexible.

Using the DATA 620/i and its Micro-EXEC capability, the user obtains the flexibility and low cost of a general purpose computer while obtaining the special purpose computer's ability to "peak" the computer's performance in particular areas using hardware subroutines.

Operation

The organization of the DATA 620/i processor allows the micro-functions to be generated by machine instructions or by an outside signal (external to the processor). Thus, machine instructions can be duplicated externally or new instructions generated. For instance, to transfer the contents of the B register to the A register requires two micro-functions -- select B register onto the S bus, and set A register.

To relinquish control of the processor to Micro-EXEC, an external control instruction is performed at the end of the Micro-EXEC sequence, the computer runs from the location specified by the P counter (which may have been changed by Micro-EXEC).

Physically the Micro-EXEC subroutine logic is located externally to the central processor, usually in an I/O chassis. As shown in Figure 9.1 Micro-EXEC logic interfaces to the I/O bus and the μ bus. This allows I/O instructions to be used in the calling sequence for a Micro-EXEC subroutine. Micro-EXEC is basically a subroutine and is handled like a software subroutine except for a few details. In some aspects, Micro-EXEC looks like a

peripheral, since it is external to the processor and interfaces to the I/O bus.

The Micro-EXEC can be entered only from the program, by the use of a external control command. (104XXX). This code will allow 9 bits for selection of the Micro-EXEC operation to be executed. Micro-EXEC Select Strobe is supplied on the Micro-bus (same timing as function ready) for use in strobing the 9 bits on the I/O cable.

The external control (104XXX) will set a flip-flop in the option card that will set inhibit clock at the trailing edge of function ready and also set a Micro-EXEC on level to the Micro bus.

The trap on shift flip-flop is set to prevent the incrementing of P and prevent the re-execution of the Micro-EXEC instruction. This is necessary to refetch the instruction that was in the memory data register. This procedure requires that the U register not be Micro-EXEC function. The computer would remain in the instruction address phase during the Micro-EXEC operation with clock enable off and as a result no CL1 and CL2 pulses will be generated.

The normal reset of trap on shift will provide the correct timing to readdress the instruction following the Micro-EXEC operation.

Micro-EXEC capability is available in several forms. The simplest form provides the customer the Micro bus (brought out to the rear panel) and the interface logic in the main frame. The Micro bus contains the 36 signals required to control the component parts of the processor. Table 9.1 lists these signals and their functions.

Or if desired, Varian Data Machines will design and implement the Micro-EXEC hardware complete with documentation.

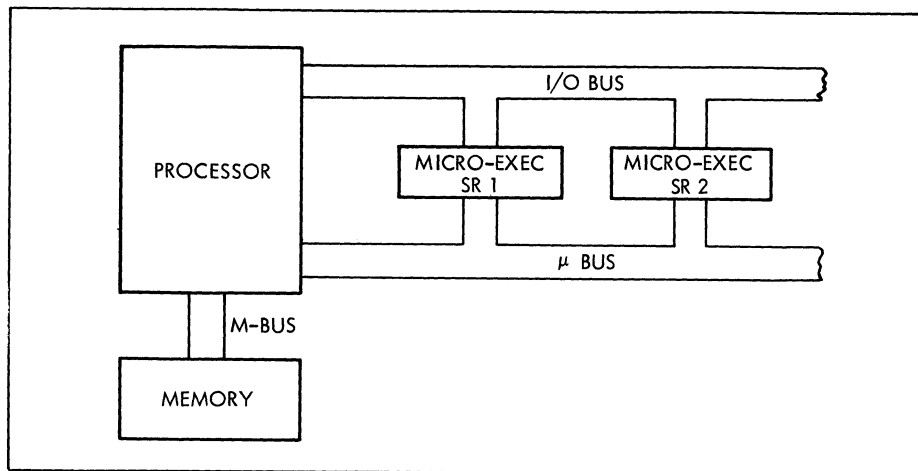


Figure 9-1. Micro-Exec Interface

Table 9-1. Signals for μ Bus

Control

1. MCLX+ - 50 nsec. at 450 nsec. (master clock) (output).
2. PHCL+ - phase clock (divide master clock by two) (output).
3. μ -Exec Select Strobe - select μ -Exec (comes on 450 nsec. before μ -Exec on) (output).
4. μ -Exec On (From Option Card) - enable customer μ -Exec (output).
5. μ -Exec Off (to Option Card) (timed with phase clock) - return to program.

Functions

1. SLAX+(level) Select A Register - gates contents of A register onto S bus.
2. SETA+(pulse) Set A Register - strobes contents of C bus into A register.
3. SLBX+(level) Select B Register - gates contents of B register onto S bus.
4. SETB+(pulse) Set B Register - strobes contents of C bus into B register.
5. SLXX+(level) Select X Register - gates contents of index register onto S bus.
6. SETX+(pulse) Set X Register - strobes contents of C bus into X register.
7. SLRX+(level) Select R Register - gates contents of R register into adder.

Table 9-1. Signals for μ Bus (Continued)

8. SLRI+(level) Select R Register Inverted - selects 1's compliments of register onto G bus.
9. SETR+(pulse) Set R Register - strobes contents of W register into the R register.
10. SETW+(pulse) Set W Register (memory data) - strobes contents of C bus into W register.
11. SETP+(pulse) Set P Register - strobes contents of C bus into P register.
12. SETL+(pulse) Set Memory Address - strobes contents of C bus into L register.
13. SLGB+(level) Select G Bus - selects G bus into adder.
14. WRTX+(level) Write (memory) - indicates a write cycle (total 1350 nsec).
15. FCXX+(level) Full Cycle (memory) - starts a full memory cycle.
16. MSCl (pulse) Memory Start Clock - strobes starts memory cycle.
17. EBDX+(level) E Bus Drive - enables E bus drivers in processor.
18. EBRX+(level) E Bus Receive - enables E bus receivers in processor.
19. SABX+(level) Select A Bus - select adder output to C bus.
20. INCR+(level) Increment - causes one to be added to data at adder.
21. ANDX+(level) And - G bus and adder output go to C bus.
22. EXOR+(level) Exclusive OR - select and G bus and not output of adder.
23. SHLX+(level) Shift Left - enables shift left gates.
24. SHRX+(level) Shift Right - enables shift right gates.
25. SGAX+(level) Shift gate A - input to adder unit (A000) for shift left.
26. SGCX+(level) Shift gate C - input to adder unit (A015) for shift right.
27. AB15+ or AB17 (level) Adder Unit Bit 15 (output).
28. CA15+ or CA17 (level) Carry Out of Adder (output).
29. AB00+(level) Adder Bit 00 (output).
30. SELP (level) Select P Register - gates contents of P register onto S bus.
31. SGBC (level) Shift Gate B.

NOTES:

1. GND is true level.
2. Levels must be on 450 nsec.
3. Pulses must be 50 nsec.

SECTION 10

DATA 620/i ASSEMBLY SYSTEM

10.1 INTRODUCTION

The DATA 620/i assembler (DAS) assists in program preparation by allowing instruction, addresses, address modifiers and constants to be specified in a straightforward and meaningful manner. Instruction mnemonics such as STB (store B register) are used in place of numeric instruction codes. Various memory locations (addresses) may be referred to by labels, rather than absolute locations. Constants may be defined without having to convert the numbers into binary or octal form. Useful comments may be added either between symbolic statements or with the symbolic statement itself to allow easy program check-out and documentation.

DAS reduces much of the tedious bookkeeping associated with machine language programming, but does not compromise the programmer's ability to fully utilize the DATA 620/i.

The basic assembly (DAS I) operates in a DATA 620/i system, which consists, as a minimum, of 4096 words of memory and an on-line teletype. The standard assembly (DAS I-F) requires 8192 words of memory.

Provisions have been made to utilize additional facilities such as high-speed paper tape, magnetic tape, card reader, card punch, additional memory and line printer if these components are available.

DAS is a two- or three-pass assembly system, which means that the source program must be

read two or three times for complete assembly. During the first pass, values are assigned to all labels appearing in the location field (page 114) and placed in the label table. During the second pass, the appropriate values for the instruction field and the variable field (page 115) are assembled into the object instruction and, together with the remarks field, are listed on the printer. During pass three, the object instructions are punched onto paper tape. In certain peripheral I/O configurations, passes two and three may be combined by the operator.

The SYMBOLIC LISTING is formatted as an 8-1/2" x 11" page with a one-inch margin at the top and bottom.

The OBJECT PROGRAMS are prepared in a standard binary format acceptable by the loader programs (section 11.6).

10.2 THE DAS SOURCE LANGUAGE

DAS translates symbolic instructions (the source program) into binary computer code (the object program). Except for certain pseudo instructions (section 10.4) each symbolic source statement will generate one or two computer word(s).

Computer codes generated by DAS fall into two categories, instructions and data. The instructions are described in section 10.3 and the data are described on pages 126 - 128.

A source statement consists of several parts, or fields. Each source statement may contain a combination of these fields depending on the requirements of the instruction or pseudo instruction being processed. The fields are: label, instruction, variable and remarks fields.

DAS Characters

The following characters are recognized by the DAS assembler:

Alphabetic characters

ABCDEFGHIJKLMNOPQRSTUVWXYZ\$

Note that the \$ is treated as an alphabetic character. It is used mainly as the first character of system software labels.

Numeric characters

0123456789

Special characters

+	(plus sign)	b	(blank)
-	(minus sign)	@	(at sign)
*	(asterisk)	[	(left bracket)
/	(slash)	]	(right bracket)
.	(period)	<	(less than sign)
=	(equal sign)	>	(greater than sign)
,	(comma)	?	(question mark)
'	(prime)	↑	(up arrow)
(	(left parenthesis)		
)	(right parenthesis)		

←	(left arrow)*	#	(pound sign)
\	(back slash)	%	(percent sign)
!	(exclamation point)	&	(ampersand)
"	(quotes)	:	(colon)
		;	(semi-colon)

Teletype control characters

CR (carriage return) LF (line feed)

Source Statement Format

The input to DAS is a sequence of statements that are entered from various input media. In any case, a statement cannot exceed 80 characters, whether they come from an 80-column card or from individual records on a continuous medium (punched tape, magnetic tape, typewriter keyboard). The 80 or fewer characters are divided into four fields: a Label, an Instruction, a Variable field and Remarks. The Variable field itself may be further divided into sub-fields for certain types of instructions.

A special statement, called a Comment statement, is treated separately by the DAS. This statement is not divided into fields.

Punched Card Format

When input is presented to DAS on punched cards, there are certain rigid formatting rules that apply. The Label must start in column 1 and not extend beyond column 6. Column 7 must be blank. The Instruction starts in

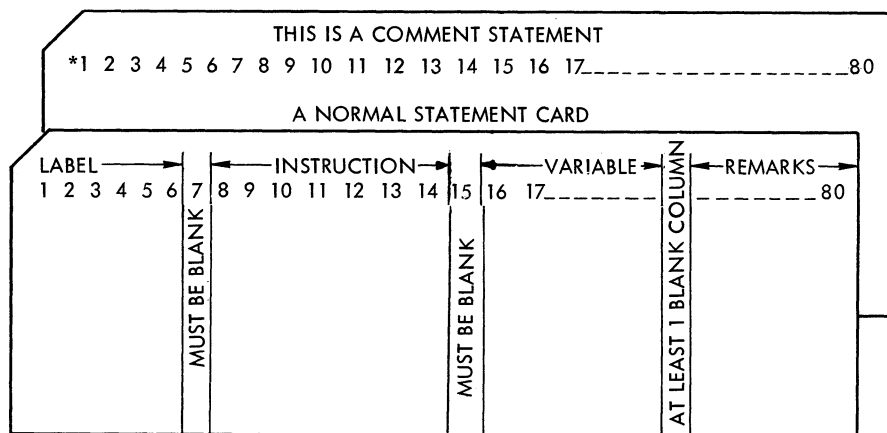
*replaced by blank on magnetic tape and punched cards.

column 8 and must not extend beyond column 14. Column 15 must be blank. The Variable field begins in column 16 and may extend to column 80, but is terminated by the first blank column (with the exception of alpha constants – see page 10-6). If column 16 is

blank, the variable field does not exist. The Remarks start anywhere after the blank column that terminates the Variable field.

A Comment statement must start with an asterisk (*) in column 1.

A COMMENT CARD



Continuous Medium Format

An alternative, column-independent, input form is provided by punched paper tape (or magnetic tape, or entry from a keyboard such as a teletype). A statement may be any length up to 80 characters plus a carriage return and line feed. The various fields and sub-fields are separated by commas with the

exception of the Remarks which are separated from the Variable field by a blank and terminated by the carriage return/line feed. Any number of blanks (or no blanks) may precede the Label, but if the first non-blank character is a comma, the Label is non-existent. If the first non-blank character is an asterisk (*), the statement is treated as a Comment.

A NORMAL STATEMENT

LABEL		INSTRUCTION		VARIABLE FIELD		REMARKS	CARRIAGE RETURN/ LINE FEED
	,		,		b		

A COMMENT

	ANY SEQUENCE OF UP TO 79 CHARACTERS TERMINATED BY	CARRIAGE RETURN/ LINE FEED
*		

Label Field

Labels consist of from one to four alphanumeric characters, the first of which is alphabetic. Special characters are not allowed in a label. Additional alphanumeric characters may be added to the first four characters of the label to form an extended label for the convenience of the programmer. However, the assembler recognizes only the first four characters. Labels are usually attached to only those source statements that are referred to elsewhere in the program, but this is not a requirement. Values are associated with the labels during the first pass of the assembler. Numeric values in the Label field are used as targets for a GOTO statement.

Instruction Field

The Instruction field contains special alphanumeric codes. These may be mnemonic codes which describe computer instructions or they may be pseudo-codes which are instructions to the assembler. An asterisk (*) following the mnemonic code indicates indirect addressing. The standard mnemonic codes and their definitions are listed in the tables in Appendix C.

The pseudo-codes are listed in table F-1 and are described in section 10.4. Additional mnemonic codes may be defined by the program through the use of the pseudo-code OPSY.

Note that the same mnemonic code may be used in the label and instruction fields without conflict.

Variable Field

The purpose of the variable field varies with the needs of the individual instruction. It may be used to define the address field of an instruction, one of the other sub-fields of an instruction, or some data constants. The variable field may consist of a label, a constant, or an expression which consists of a combination of labels and constants in which the constants must be integers 2^{15} (2^{17}). The expressions that may be used in the DAS assembly system are similar to arithmetic expressions, except that no parentheses may be used. The following arithmetic operators are available in the variable field of DAS:

- + (addition)
- (subtraction)
- * (multiplication)
- / (division)

All arithmetic operations are performed in the integer mode, i.e., modulo 2^{15} . The expression $A+B/C*D$ is equivalent to the algebraic expression $A+(B/C)*D$. The operations are performed from left to right with the multiply and divide operations taking

) $\pm$ integer. fraction E $\pm$ exponent

Examples:)375.64E+7,)9.E-2,).1E+12,)-4.+20,)7.53,)-.002

A right parenthesis, digit and decimal point must be present. All other items are optional.

precedence over the add and subtract operations.

Access to the current value of the location counter may be gained by the use of an asterisk (*), when used as the first character of the variable field. An asterisk immediately preceding an operator is treated as the location counter rather than an operator. Thus, the expression $*+1$ is interpreted as meaning the current value of the instruction counter plus one.

Constant-generation facilities available in the DAS assembly system are described below.

Decimal integers. A decimal integer is an optionally signed string of from one to six digits, the first of which is not zero.

Examples: 1,7,-3,+327

Octal integers. An octal integer is an optionally signed string of from one to seven octal digits, the first of which is always zero.

Examples: 07,-044,+014

Floating-point numbers. Floating-point numbers can be assembled by DAS in the following form:

The format of the assembled data is shown below.

16-Bit Format

		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
L	S	Exponent + 0200										High Mantissa					
L+1	0	Low Mantissa															

18-Bit Format

		17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
L	S	Exponent + 0200										High Mantissa							
L+1	0	Low Mantissa																	

The sign bit of the second word is always set to zero.

Negative data are in 1's complement form in the first word, the second word is always in true (absolute value) form.

Alpha constant. An alpha constant is a string of characters enclosed by primes (''). An alpha constant is represented internally as an 8-bit ASCII code. When one character is generated, the character is right-justified with leading zeros. Each memory location may contain two characters. A blank in the string is recognized as a character, but does

not terminate the variable field as long as it is between the primes (').

An alpha constant may be a term in an arithmetic expression. If more than one word is generated by the constant, only the last word generated is operated on arithmetically.

Example: 'A'*0400, 'AB'+1, 'ABCD'+1

(on the last example, two words would be generated, with a one added to the second word.)

Some examples of words generated by character constants are given below:

	17 15		8 7								0							
'A'	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1

One word is generated.

0

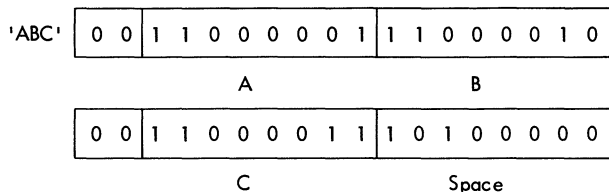
A

'AB'	0	0	1	1	0	0	0	0	0	1	1	1	0	0	0	0	1	0
------	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

One word is generated.

A

B



Two words are generated. Note that the space character code is used to fill the low-order eight bits of the second word if an odd number (except 1) of characters is specified within the primes.

If the DATA 620/i has an 18-bit word length, zeros are generated in bits 16 and 17 of each word.

Address constant. An address constant consists of a label, number or expression enclosed in parentheses, and generates a 15-bit address with bit position 15 set to a logical zero to indicate a direct address.

Examples: (A+2), (3), (A), B+4, 5, C

A is an address symbol and its value is obtained from the Label table. If the program is relocated, the value of the address constant is changed to agree with the location assigned to the word labeled A. Note that the parentheses around the address constants are optional in the above examples.

Indirect address constant. An indirect address constant consists of an address constant

followed by an asterisk (*), and generate a 15-bit address with bit position 15 set to a logical one to indicate an indirect address.

Examples: (A)*, (A+3)*, (3)*, (2*A+3)*

Note that the parentheses are mandatory here to distinguish between the asterisk as a multiply symbol and as an indirect symbol.

Literals. Literals allow the programmer to refer to a constant in the variable field and have DAS generate the data and assign a location in memory. Even though a literal may be used many times, only one location will be generated for any given literal.

A literal reference is indicated by an equal sign (=) followed by any format of a one-word constant.

Examples: =3, =+3, =-3, =044, =(A+2)*, ='A', ='GO'

For certain instructions, more than one expression is desired. In these cases the expressions are separated by commas (,).

Note that the expressions deal with the values assigned to labels, and not the contents of memory locations that may be referenced by the labels.

Remarks Field

The remarks field is separated from the variable field by at least one blank character. Therefore there must be no blanks in the variable field except within Alpha constants enclosed by primes (''). The information in the remarks field is ignored by the DAS

assembler and the programmer may put in any comments that help him in documentation and debugging.

10.3 DATA 620/i INSTRUCTIONS

The following paragraphs assume the 16-bit configuration of the DATA 620/i. Each of the four instruction types is described in the following paragraphs. Optional Instructions

are recognized only when installed in the object computer. A list of the instruction mnemonics acceptable to the assembler is given in Appendix C.

Type-1 Instructions

Type-1 instructions occupy one computer word and are addressable. DAS recognizes the following forms:

Label Field (optional)	Instruction Field	Variable Field	Comments
Label	Mnemonic	Expression	The expression value is the effective address.
Label	Mnemonic	Exp 1, Exp 2	The value (modulo 512) of expression Exp 1 is added to the contents of the X register or the B register to form the effective address. The expression Exp 2 must have a value of 1 or 2 to designate the X or the B register, respectively.
Label	Mnemonic*	Expression	The expression value is the indirect address of the operand.
Label	Mnemonic	(Expression)*	

If the first form of the instruction listed above is used, DAS will choose the addressing mode of the generated computer instruction according to the following rules:

- a. If the specified address lies within core locations 0-2047 inclusively, the direct address will be used.

- b. If the specified address lies outside core locations 0-2047 but not more than 512 and not less than one word beyond the current instruction, the mode of addressing is relative to the location counter.

- c. If neither condition a nor condition b is true, a 15-bit address will be

generated in memory area 0-511 (called bank 0), and the bank-0 address will be used in the instruction in the indirect mode.

Type-1 mnemonics recognized are: (see Appendix C and table C-3)

LDA (load A register)
LDB (load B register)
LDX (load X register)
STA (store A register)
STB (store B register)
STX (store X register)
ADD (add to A register)
SUB (subtract from A register)

INR (increment memory word)
ERA (exclusive-OR to A register)
ORA (inclusive-OR to A register)
ANA (AND to A register)
MUL (optional multiply)
DIV (optional divide)

Type-2 Instructions

Type-2 instructions require two computer words. The second word is the direct or indirect address if the instruction is a jump, jump-and-mark or execute. The second word of an immediate instruction is the operand. The second word of the extended-address instruction is the operand address. DAS recognizes the following forms:

Label Field (optional)	Instruction Field	Variable Field	Comments
Label	Mnemonic	Expression	The expression value is the effective jump, jump-and-mark or execute address, or it is the operand of an immediate instruction.
Label	Mnemonic*	Expression	The expression value is the indirect jump, jump-and-mark or execute address.
Label	Mnemonic	(Expression)*	This is exactly equivalent to the above form.

The following type-2 mnemonics are recognized as Immediate instructions: (see Appendix C and table C-9)

LDAI STXI ORAI
LDBI ADDI ANAI
LDXI SUBI DIVI (optional)
STAI INRI MULI (optional)
STBI ERAI

The following type-2 mnemonics are recognized as jump, jump-and-mark and execute instructions: (see Appendix C and table C-7)

JMP JBZ JMPM
JOF JXZ JOFM
JAN JSS1 JANM
JAP JSS2 JAPM
JAZ JSS3 JAZM

JBXM	XEC	XBZ
JXZM	XOF	XXZ
JS1M	XAN	XS1
JS2M	XAP	XS2
JS3M	XAZ	XS3

Type-3 Instructions

Type-3 instructions are two-word computer instructions with a direct or indirect address in the second word. They differ from the type-2 instructions in that the variable field of the symbolic instruction contains two sub-fields instead of one.

DAS recognizes the following forms of type-3 instructions:

Label Field (optional)	Instruction Field	Variable Field
Label	Mnemonic	Exp 1, Exp 2
Label	Mnemonic*	Exp 1, Exp 2
Label	Mnemonic	Exp 1, (Exp 2)*

DAS recognizes the following type-3 mnemonics:

Dummy conditional jumps:

JIF (jump if...)

JMIF (jump-and-mark if...) or

JIFM (jump-and-mark if...)

XIF (execute if...)

The value of the expression Exp 1 specifies which of the conditions will cause a jump, jump-and-mark or execute instruction. The

conditions of Exp 1 have the following values:

If OFLO set:	0001 ₍₈₎
If A < 0:	0004 ₍₈₎
If A ≥ 0:	0002 ₍₈₎
If A = 0:	0010 ₍₈₎
If B = 0:	0020 ₍₈₎
If X = 0:	0040 ₍₈₎
If SS1 set:	0100 ₍₈₎
If SS2 set:	0200 ₍₈₎

Compound conditions may be specified by adding together the values of the desired conditions.

For example:

Instruction Field	Variable Field
JIF	0222, ALFA

Where 0222 = 0200 + 020 + 02 means: take the next instruction from address ALFA, if and only if, all three of the following conditions are true:

The A register contains a positive number: 0002

The B register contains zero: 0020

Sense switch 2 is set: 0200

The value of the expression Exp 2 is a direct or indirect jump, jump-and-mark or execute address.

The following type-3 mnemonics are recognized as extended address instructions (optional): (see Appendix C and table C-8)

LDAE	STXE	ORAE
LDBE	ADDE	ANAE
LDXE	SUBE	DIVE
STAE	INRE	MULE
STBE	ERAE	

Type-3 instructions also include the following I/O instructions: (see Appendix C and table C-10)

SEN (sense for state of an I/O device)

IME (input to memory)
OME (output from memory)

The value of the expression Exp 1 in the variable field of the instruction is the device subcode.

The value of the expression Exp 2 is a direct or indirect memory address.

Type-4 Instructions

Type-4 instructions are single-word instructions which do not refer to a memory location. DAS recognizes the following formats:

Label Field (optional)	Instruction Field	Variable Field	Comments
Label	Mnemonic		Variable field is blank.
Label	Mnemonic	Expression	The value of the expression specifies either source/destination registers and overflow conditions, a shift count, an I/O device or function, or a halt number.

DAS recognizes the following type-4 mnemonics: (see Appendix C and Tables C-4 and C-6).

TZA, TZB, TZX	(clear register)
IAR, IBR, IXR	(increment register)
DAR, DBR, DXR	(decrement register)
CPA, CPB, CPX	(complement register)
TAB, TBA, TAX, TXA, TBX, TXB	(register transfer)
AOFA, AOFB, AOFX, SOFA, SOFB, SOFX	(overflow arithmetic)
SOF, ROF	(overflow control)
HLT, NOP	(control)

The following instruction mnemonics are functionally the same as the preceding

register change instructions except that these mnemonics allow the user to specify

multiple-source and/or destination registers, or specify whether or not function execution is dependent on the overflow conditions:

Instruction Mnemonic	Instruction Function
MERGE	Take the inclusive-OR of the contents of all specified source registers and deliver the result to each of the specified destination registers.
COMPL	Like MERGE, except the result is ones-complemented before delivery.
INCR	Like MERGE, except +1 is added to result before delivery.
DECR	Like MERGE, except +1 is subtracted from the result before delivery.
ZERO	Zero each of the specified destination registers.

The value of the expression used in the variable field of the instruction is interpreted by DAS as having the following meaning: (see table C-6(a)).

If bit 0 = 1: A is a destination register.

If bit 1 = 1: B is a destination register.

If bit 2 = 1: X is a destination register.

If bit 3 = 1: A is a source register.

If bit 4 = 1: B is a source register.

If bit 5 = 1: X is a source register.

If bit 8 = 1: The function is to be performed if and only if the overflow flip-flop is set to 1.

For example:

Instruction Field	Variable Field
COMPL	0235

where

$$0235 = 0200 + 020 + 010 + 04 + 01$$

means: take the logical INCLUSIVE-OR of the A and B registers, complement this, and put the result in both the A and X registers. Note that the result is the logical NOR of the initial A and B.

The shift mnemonics recognized by DAS are listed below and in table C-5. The expression value represents the number of positions to be shifted. A value outside the range of 0-31 is reduced modulo 31 and an error code is printed.

LSRA	(logical shift right, A)
LRLA	(logical rotate left, A)
LSRB	(logical shift right, B)
LRLB	(logical rotate left, B)
ASRA	(arithmetic shift right, A)
ASLA	(arithmetic shift left, A)
ASRB	(arithmetic shift right, B)
ASLB	(arithmetic shift left, B)
LLSR	(long logical shift right)
LLRL	(long logical rotate left)
LASR	(long arithmetic shift right)
LASL	(long arithmetic shift left)

The following single-word input/output instructions are recognized by DAS: (see Appendix G-11)

EXC or SEL	(external control)
INA	(input ORed into A)
INB	(input ORed into B)
INAB	(input ORed into A and B)
CIA	(input replaces A)
CIB	(input replaces B)
CIAB	(input replaces both A and B)
OAD	(output from A)
OBR	(output from B)
OAB	(output inclusive OR of A and B)

The value of the expression determines the I/O function and device in the case of EXC or SEL, or just the device in the other cases (the function is determined by the mnemonic instruction code). Appendix D shows the codes assigned to the various functions and devices.

For example:

Instruction Field	Variable Field
CIB	030

Will clear B and read into it from a card reader.

10.4 DAS PSEUDO INSTRUCTIONS

The following set of pseudo instructions is provided to allow the DATA 620/i programmer

complete control of the assembly process. The pseudo instructions are divided into the following groups:

- Label Definition
- Instruction Definition
- Location Counter Control
- Data Definition
- Memory Storage Reservation
- Conditioned Assembly
- Assembler Control
- Subroutine Control
- List and Punch Controls

A list of the pseudo codes is presented in Appendix F.

Label Definition

The label table is a list of labels that occur in the source program. To each label, there is a corresponding value, usually an address. The programmer may assign arbitrary values to labels by means of the pseudo instructions described in the following paragraphs.

EQU pseudo instruction.

Label field:	a label.
Instruction field:	EQU.
Variable field:	An expression.

The label is placed in the label table and assigned the value of the expression in the variable field. If the label is already in the label table, an error message (DD) is printed and the value of the expression replaces the value in the table. Any label appearing in the expression must have been defined previously for correct assembly.

SET pseudo instruction.

Label field:	a label.
Instruction field:	SET.
Variable field:	an expression.

If the label in the location field has not yet appeared in the location field in any instruction, this label is entered into the label table and assigned the value of the expression in the variable field. If the label is already in the label table the value of the expression in the variable field replaces the previous value of the label. Any labels appearing in the variable field must have been defined previously for correct assembly.

EQU and SET are essentially the same except that redefinition of a label is permitted by SET without an error message.

MAX pseudo instruction.

Label field: a label.

Instruction field: MAX.

Variable field: two or more expressions separated by commas.

The label in the location field is assigned the greatest of the algebraic values of the expressions appearing in the variable field. All labels appearing in the variable field must have been previously defined for correct assembly.

MIN pseudo instruction.

Label field: a label.

Instruction field: MIN.

Variable field: two or more expressions separated by commas.

The label in the location field is assigned the smallest of the algebraic values of the

expressions appearing in the variable field. All labels appearing in the variable field must have been previously defined, for correct assembly.

Instruction Definition

The DATA 620/i programmer may redefine a standard instruction mnemonic with the pseudo instruction OPSY.

OPSY pseudo instruction.

Label field: a label.

Instruction field: OPSY.

Variable field: an instruction mnemonic.

The label becomes an instruction mnemonic, with the same definitions as the mnemonic in the variable field. This pseudo instruction is used to define new instruction mnemonics to be used in the current assembly.

Example: CLA OPSY LDA
 CLA BETA

The CLA BETA will be assembled as though it had been LDA BETA. All succeeding uses of the op-code CLA will be treated as though they were LDA.

Location Counter Control

Five pseudo instructions are provided for controlling the DAS location counters (LC). Multiple location counters are provided in the DAS assembler, along with pseudo instructions to preset or modify the values of an individual location counter.

The following table lists the five LC labels which are standard in the DAS system. These LC labels need not be created by the DATA 620/i programmer.

LC Label	Initial Value	Intended Use
SYOE	00001 ₍₈₎	Controls the assignment of locations to any system parameters desired by the user.
IAOR	00100 ₍₈₎	Controls the assignment of locations to indirect pointers.
LTOR	01000 ₍₈₎	Controls the assignment of locations to literals.
COMN	02000 ₍₈₎	Controls the assignment of locations within an interface area which is common to two or more programs.
Blank	04000 ₍₈₎	The blank location counter is used initially by DAS for assigning locations. This is the counter normally in use by DAS unless the programming tells it to do otherwise with USE pseudo instruction.

In addition, to the five standard location counters, the DAS programmer may create up to eight of his own location counters. This allows the programmer to create complex relocatable and overlay programs within a single assembly.

At the beginning of an assembly, there are no created location counters. DAS uses, at any time, three location counters for location assignment. The IAOR and LTOR location counters are always in use. A third location counter is used to assign locations to generated instructions and to generated data (except literals and indirect pointers). The blank location counter is initially used by DAS to control this function until another LC symbol is so designated by the pseudo instruction USE.

For a straightforward program which used one LC, complete control over the LC is maintained by pseudo instructions ORG and LOC.

ORG pseudo instruction. The location counter that is currently in use is set to the value of the expression in the variable field. If a label appears in the location field, the label is set to the value in the variable field. If a label appears in the expression, the label must have been previously defined for correct assembly.

Label field: label or blank.

Instruction field: ORG.

Variable field: an expression.

LOC pseudo instruction. The LOC pseudo instruction causes instructions and/or data following LOC to be generated as if the ORG pseudo instruction had been used to change the current LC value. However, the

value of the LC is not changed by the LOC pseudo instruction and the instructions and/or data generated are located in memory at the LC address.

The LOC pseudo instruction is used if the instructions and data following the LOC address are to be moved to the LOC address by the object program before execution. If a label appears in the variable field, the label should have been previously defined for correct assembly.

The LOC pseudo instruction may not be used with a relocatable program.

Label field: label or blank.

Instruction field: LOC.

Variable field: an expression.

BEGIN pseudo instruction. The BEGIN pseudo instruction allows the DAS programmer to create a new location counter or to redefine the value of any location counter before using it. The location counter is given a value equal to the expression in the variable field. BEGIN does not have any effect on the location counter currently being used.

Once a location counter has been used by a DAS program for location assignment, the value of that location counter may not be redefined by the BEGIN pseudo instruction. If a label appears in the expression in the variable field, the label must have been previously defined for correct assembly.

Label field: label.

Instruction field: BEGIN.

Variable field: an expression.

USE pseudo instruction. The USE pseudo instruction causes DAS to use the location counter designated in the variable field to assign locations to the instructions and data (except literal and indirect pointers) following USE.

Label field: blank.

Instruction field: USE.

Variable field: blank, COMM, SYOR or a created LC label.

If the variable field is the character string PREV, then the LC used previously is recalled. Only one previous usage is remembered. Thus, the sequence

USE A	or	USE C
USE B		USE A
USE PREV		USE B
		USE PREV
		USE PREV

are both equivalent to USE A.

Data Definition

DATA pseudo instruction. A data item may be a decimal or octal integer, a floating-point number, an alpha constant, a direct or indirect address or an expression. Several data items may be generated with a single DATA instruction. They will be assigned successive memory locations in the order in which they appear.

If a label appears in the location field, the label is assigned to the memory location of the first generated word.

Label field: a label or blank.

Instruction field: DATA.

Variable field: One or more data items separated by commas.

Example: DATA 13,075,)7.5, 'ONE', LOC, (LOC+7)*

will result in 8 words of memory being given values, the location of the first of these words will be the value assigned to the label DATA.

PZE pseudo instruction. The PZE (plus zero) pseudo instruction is essentially the DATA pseudo instruction except that the sign bit of the data word is always set to zero (plus).

Label field: a label or blanks.

Instruction field: PZE.

Variable field: one or more data items separated by commas.

MZE pseudo instruction. The MZE (minus zero) pseudo instruction is essentially the DATA pseudo instruction except that the sign bit of the data word is set to one (minus).

Label field: a label or blanks.

Instruction field: MZE.

Variable field: one or more data items separated by commas.

Memory Storage Reservation

BSS pseudo instruction. BSS causes the location counter to be increased by the value of the expression in the variable field. If a label is used, it will be assigned the value that the location counter had before it was increased. This essentially sets aside a block of memory whose length is equal to the value

of the expression. The label takes the value of the location of the first word in this block.

Label field: a label or blanks.

Instruction field: BSS.

Variable field: an expression.

BES pseudo instruction. BES causes the location counter to be increased by the value of the expression in the variable field. If a label is used, the label is assigned to the address value of the incremental location counter minus one. This is similar to the BSS except that the label refers to the last word in the block.

Label field: a label or blanks.

Instruction field: BES.

Variable field: an expression.

DUP pseudo instruction. DUP is used to produce multiple copies of 1, 2 or 3 source statements.

Label field: blank.

Instruction field: DUP.

Variable field: one of three forms, as follows:

Form 1: No address fields. The instruction is ignored.

Form 2: One address field. Example: DUP, n (the next source statement is duplicated n times).

Form 3: Two address fields. Example, DUP, n, m (the next m source statements are duplicated n times where $m \leq 3$, $n \leq 32,767$.) If either field contains a zero the field will be treated as though a one were presented.

Conditional Assembly

The following five pseudo instructions are provided to conditionally assemble various portions of a DATA 620/i program.

IFT and IFF pseudo instructions.

Label field: blank.

Instruction field: IFT or IFF.

Variable field: one, two, or three expressions separated by commas. IFF (if false) is the logical complement of the IFT (if true) instruction.

The instruction,

Instruction Field	Variable Field
IFT	A, B, C

means: include the next line of code if $A < B$ and $B \leq C$. In the variable field, the form A,,B means $A \neq B$. The form A is true if $A \neq 0$, otherwise false.

The following are examples of frequently used forms:

Instruction Field	Variable Field	Comments
IFF	A,,B	for $A = B$
IFT	A, B, B	for $A < B$
IFT	0, A, B	for $A < B$ and $A > 0$
IFF	A	for $A = 0$

GOTO pseudo instruction. GOTO is used to skip more than one instruction. GOTO, which usually follows an IFT, or IFF pseudo instruction, may not be used to skip to an earlier point in the program. All instructions following GOTO, up to but not including the first instruction containing the designated target in its label field, are skipped.

The instructions that have been skipped are listed, unless suppressed by the SMRY pseudo instruction (page 10-131) or by a comma following the target in the variable field. In no case are the skipped instructions assembled into object code.

Label field: blank.

Instruction field: GOTO.

Variable field: target in one of the forms:

- a) symbol
- b) symbol,
- c) integer
- d) integer,

CONT and NULL pseudo instructions. The CONT (continue) or NULL pseudo instruction may be used to provide a target for a previously appearing GOTO. No object data is generated with the CONT or NULL pseudo instructions. The CONT instruction will not be listed if the SMRY pseudo instruction is in effect.

Label field: target (symbol or integer).

N	EQU	16
	IFT	N-16
	GOTO	YYY
	(CODING FOR 16 BIT)	
	IFF	N-16
	GOTO	123
YYY	(CODING FOR 18 BIT)	
123	CONT	

Note that the label used with the NULL instruction will be included in the symbol table. The label used with the CONT instruction will be ignored.

Assembler Control

END pseudo instruction. DAS requires the END pseudo instruction as the last source statement in the program. The value of the expression in the variable field is used by the loader as the entry point into the program, after the program has been loaded into the DATA 620/i. A blank expression field designates location 00000 as the entry point.

Label field: blank.

Instruction field: END.

Variable field: an expression.

Instruction field: CONT or NULL.

Variable field: blank.

Note that the target may be a numeric value rather than an alphabetic symbol.

Example:

N-16=0 (FALSE)
GO INCLUDE CODING FOR 18 BIT

N-16=0 (FALSE)
BY PASS 18 BIT CODING

COMMON CODING

MORE pseudo instruction. MORE is used to inform DAS that additional inputs are to be placed in the source input device. The DAS assembly system executes a halt to allow the additional source statements to be placed in the input device. Assembly resumes when the RUN pushbutton on the computer control console is pressed. This pseudo instruction is never listed.

Label field: blank.

Instruction field: MORE.

Variable field: blank.

Subroutine Control

The three pseudo instructions provided for the creation and use of closed subroutines are described in the following paragraphs.

ENTR pseudo instruction. ENTR causes DAS to assemble a closed subroutine. The label in the location field is the name of the subroutine. The ENTR generates the lineage word (zero) in the object subroutine.

Label field: label.

Instruction field: ENTR.

Variable field: blank.

RETU pseudo instruction. RETU is used to return from a closed subroutine. An unconditional branch is generated to the value of the expression in the variable field.

Label field: label or blank.

Instruction field: RETU.

Variable field: an expression.

CALL pseudo instruction. If a label appears, the label is entered into the label table and assigned the present value of the (current) location counter. The first subfield must contain a valid label (the name of a subroutine). The list subfields may contain any valid DATA items.

Label field: a label or blank

Instruction field: CALL.

Variable field: one or more subfields, as follows:

a) symbol (required)

b) parameter (optional)
list

c) error re- (optional).
turn list

Example:

,CALL, FUNC, X, Y + 1, ERR, (GOOF)*

This produces a machine code identical to that which would be obtained by:

,JMPM, FUNC
,DATA, X, Y + 1, ERR, (GOOF)*

List and Punch Controls

The following eight pseudo instructions provide the DATA 620/i programmer complete control over the listing and punching functions during program assembly. These controls are operative only during the second pass of DAS.

LIST pseudo instruction. LIST informs the DAS assembly system that a program listing is to be produced. DAS is initially in a LIST condition.

Label field: blank.

Instruction field: LIST.

Variable field: blank.

NLIS pseudo instruction. NLIS suppresses further listing of the program.

Label field: blank.

Instruction field: NLIS.

Variable field: blank.

PUNC pseudo instruction. The PUNC pseudo instruction starts the punching of an object

paper tape program from the DAS assembly system. DAS is initially in a PUNC condition.

Label field: blank.

Instruction field: PUNC.

Variable field: blank.

NPUN pseudo instruction. NPUN suppresses punching of an object paper tape from the DAS assembly system. PUNC and NPUN may be interspersed to punch only selected portions of an object program.

Label field: blank.

Instruction field: NPUN.

Variable field: blank.

SPAC pseudo instruction. The listing device is spaced by the number of lines in the variable field. The SPAC pseudo instruction itself is not listed.

Label field: blank.

Instruction field: SPAC.

Variable field: an expression.

EJEC pseudo instruction. The EJEC pseudo instruction causes the listing device to start a new page. EJEC itself does not appear on the listing.

Label field: blank.

Instruction field: EJEC.

Variable field: blank.

SMRY pseudo instruction. SMRY suppresses the listing of source statements which have been skipped by the condition assembly controls (page 10-128), and the listing of the symbol table on pass 1.

Label field: blank.

Instruction field: SMRY.

Variable field: blank.

DETL pseudo instruction. DETL removes the effect of the SMRY pseudo instruction. That is, all source statements are listed. The normal mode of operation of the DAS system is the DETL mode.

Label field: blank.

Instruction field: DETL.

Variable field: blank.

READ pseudo instruction. DAS is initially set to process up to 80 characters per line. This instruction will permit less than 80 characters from each source line to be processed by the assembler. If n is less than 20 or greater than 80, the number of characters read will be reset to 80 and a SZ message will be listed. A SMRY pseudo instruction will suppress the listing of READ statements during pass 2, unless there is a size error message.

Label field: blank.

Instruction field: READ.

Variable field: n, c.

For card input: DAS is initialized to 026 keypunch codes. The value of c may be 29 to indicate the use of 029 keypunch codes.

Instruction Field	Variable Field	Action Initiated
READ	80,29	Reads 80 columns of 029 [†] codes, in all succeeding cards.
READ	72,26	Reads 72 columns of 026 codes, in all succeeding cards.
READ	,29	Does not change number of columns read, does change type of codes.
*READ	80	Reads 80 characters, does not change codes.

*for tape input, C is not used, since only teletype code is recognized. The READ pseudo instruction would rarely be used with the tape input, since the tape format is flexible without it.

[†]029 codes refer to EBCDIC codes.

If the code type is not 26 or 29 the assembly will stop with A, B, X and U registers equal to 26. At this time the card may be corrected and put back in the card reader. Pressing the RUN button will continue the assembly.

10.5 DAS OUTPUT LIST

DAS Source Listing

The DAS assembly system allows the programmer to obtain an on-line listing of his program, either in parts, or the entire program,

as the program is being assembled. The symbolic (source) program and the object (absolute) program are listed side-by-side on the listing device (either teletype or printer).

Error analysis is performed during assembly and, as errors are detected, error codes (section 10.6 and Appendix F), are printed on the line following the source/object information.

The list controls pseudo instructions: LIST, NLIS, SPAC, EJEC, SMRY, and DETL are described on pages 130 and 131.

The format of the data on the output listing is:

Location	Object Code	Address Mode	Source Statement	Comments
014000			, ORG , 014000	
014000	000000		ABS , ENTR ,	
014001	001002		, JAP* , ABS	
014002	114000	R		
014003	005211		, CPA ,	
014004	001000		, JMP* , ABS	
014005	114000	R		
	000000		, END ,	

Address modes include:

- C: FORTRAN common reference.
- E: externally defined.
- I: indirect pointer.
- R: absolute.

Up to four error messages may occur on a line of output listing. The error message is preceded by a list of the source statement. The error codes produced by DAS are shown in Appendix F.

10.6 DAS ERROR MESSAGES

The DAS assembly system performs extensive syntax checking during both passes of the assembler. During the first pass, detectable errors are listed. When an error is detected on the second pass of DAS, the following information is listed:

Error code

Value of location counter

Object code when instruction has been assembled unless a NLIS pseudo instruction is in effect or a list suppress comma is present on a GOTO pseudo instruction.

10.7 OPERATING THE DAS ASSEMBLY SYSTEM

The assembler tape is loaded into memory using the binary load program (see section 11.6). After the assembler loading is complete the normal system input, output, and listing devices are readied, the SENSE switch(es) are set depending on pass. SENSE switch 1 for pass 1 and SENSE switches 2 and 3 for pass 2. To begin assembly, RUN at location 000001.

Termination of pass 1 and 2 is initiated whenever an END pseudo-op is detected. END causes a HALT 0777 to be executed with the A, B and X registers set to -1 (all ones). To initiate pass 2, reset the I/O

devices, set the SENSE switches, and RUN. Pass 2 may be repeated as often as desired to produce extra copies of the program.

The computer will execute a HALT 0777 when a MORE pseudo-op is detected, and display 0170017 (octal) in the A, B and X registers. Prepare the input units and RUN. Synchronization errors are detected on pass 2 when the address value of a label does not agree with the value assigned on pass 1. Synchronization errors are due to misreads of the source tape and cause DAS to halt with the A, B and X registers set to 0777. To continue the assembly process, press RUN. The assembler will reset the location counter to the value assigned during the first pass, print the error message SE and continue.

10.8 FORTRAN PSEUDO INSTRUCTIONS

General

The following special op codes are provided for the DATA 620/i programmer in order to provide assembly output compatible with the FORTRAN loader.

FORT op code. This op code must be the first line of code in an assembly, except for comments. It indicates that the output must be compatible with the FORTRAN relocatable loader.

NAME op code. This op code must be the second line of code in an assembly, except for comments. It contains the name of the entry point in the address field. The label field is left blank. The name indicated is provided to the assembly program and output for the loader in order to allow linkage to the routine from other routines. Multiple entry points are allowed.

COMN op code. This op code is used to define common areas. The area name is placed in the label field and the length in the address field. This op code may be placed anywhere within the program. Only one name is defined for each use of the op code, and the names and area lengths are cumulative. It has approximately the same effect as a series of BSS instructions, except the area is defined to be in the common pool.

EXT op code. This op code is used to indicate that a symbol is not undefined, but resident in another routine. The symbol to be so identified is placed in the label field. The variable field is unused. One such symbol can be defined with each use of this op code. This code may be placed at any point within the program.

Relocation

In order to allow relocation, the system requires that all one word instructions that address locations in memory use the relative forward method of addressing. All two word instructions are legal.

Literals

No literals may be used. The use of immediate instructions is recommended.

Restrictions

All expressions containing symbols defined with COMN or EXT instructions must be the second word of two word instructions, or part of a DATA, PZE or MZE instruction.

The FORTRAN compiler uses two words for each value retained in core. For this reason it is necessary for the assembly language programmer to make allowance for this when defining COMMON.

FORTRAN had the statement: COMMON
(4), B(3,4) DAS should have:

```
A , COMN,4*2  
B , COMN,3*4*2
```

Modes

All symbols and expressions are given a mode. This mode is either external, common, relative or absolute. The definition of the mode is assigned by the assembler according to certain rules. Both symbols and expressions have a mode. The mode of an expression is determined by the mode of the symbols used within the expression.

The mode of a symbol is defined as follows:

If the symbol is defined with the EXT op code, the mode is E.

If the symbol is defined with the COMN op code, the mode is C.

If the symbol is a numeric constant, the mode is A.

If the symbol is * used as the current location, the mode of the * is R.

If the symbol is defined by an EQU, SET etc., the mode is that of the expression on the right side of the op code.

If the symbol is a label in a program, the mode is R.

Certain restrictions appear within the DAS assembler when providing FORTRAN compatible output. The restrictions on expressions are:

No expression may contain both mode E and C symbols.

Any type E expression must consist only of the type E symbol.

No type E, C or R expression should include the multiplication or division of a type E or C symbol.

No expression should contain the sum or difference of a mode C symbol and a mode R symbol, or a mode E symbol and a mode R symbol.

No expression should contain the sum of two mode E, C or R symbols.

A mode A symbol may be added to or subtracted from a mode C or R symbol.

The mode of an expression is assigned as follows:

If the expression contains any symbol of mode E, the expression is mode E.

If the expression contains any symbol of mode C, the expression is mode C.

If the expression contains only mode A symbols, the expression is mode A.

If the expression contains A and R symbols, the mode is R if an odd number of mode R symbols appear, otherwise the mode is A.

Examples of legal and illegal expressions:

EEEE	,	EXT	,		EEEE defined as type E
CCCC	,	COMN	,	6	CCCC defined as type C
RTN	,	ENTR	,		RTN is type R, a label
TBL	,	BSS	,	50	TBL is type R
ABL	,	BSS	,	'A'+5	ABL is type R
LENG	,	EQU	,	*-TBL	LENG is type A, length of area
	,	CALL	,	EEEE, TBL, LENG	
	,	LDA	,	*+6	Ok, relative forward
	,	LDA	,	CCCC+6	Illegal, one word inst, not R or A
	,	LDXI	,	CCCC+6	Ok, two word instruction
	,	LDA	,	0,1	Get CCCC+6 to A, legal
	,	DATA	,	EEEE+4	Illegal
	,	DATA	,	CCCC+4	Legal
	,	DATA	,	CCCC+LENG	Legal
	,	DATA	,	TBL+LENG-5	Legal, mode is R

Example of FORTRAN Compatible Assembly

```

000000      R      ,  FORT      ,
000017      R      ,  NAME     ,  $PE
000000      E      $SE      ,  EXT      ,
000000      E      $QS      ,  EXT      ,
000000      E      $QE      ,  EXT      ,
000000      074025      ,  STX      ,  $PE+7
000001      034021      ,  LDX      ,  $PE+4
000002      054025      ,  STA      ,  $PE+9
000003      064025      ,  STB      ,  $PE+10
000004      015000      ,  LDA      ,  0,1      PAR.1
000005      034020      ,  LDX      ,  $PE+7
000006      002000      ,  JPM      ,  $QS
000007      E      000000
000010      R      ,  DATA     ,  $PE+7
000011      014016      ,  LDA      ,  $PE+9
000012      024016      ,  LDB      ,  $PE+10
000013      002000      ,  JPM      ,  $QE      A**B SUBROUTINE
000014      E      000000
000015      R      ,  DATA     ,  $PE+7
000016      001000      ,  JMP      ,  0
000017      000000
000017      ,  ORG      ,  *-1
000017      000000      $PE      ,  ENTR      ,
000020      002000      ,  CALL     ,  $SE, 1
000021      000000      E
000022      000001
000023      ,  DATA     ,  0
000024      001000      ,  JMP      ,  *-20
000025      000000      R
000026      000000      ,  DATA     ,  0,0,0,0
000027      000000
000030      000000
000031      000000
000000      ,  END      ,

```


SECTION 11

DATA 620/i SUBROUTINES

11.1 INTRODUCTION

This section is intended to acquaint the programmer with the standard subroutine library and how it is used. It is divided into the following areas:

- a. Programmed Arithmetic
- b. Elementary Functions
- c. Utility and Debugging Routines
- d. Executive Routines

Each routine is documented in accordance with the programming standards set forth in the following pages. These standards show the various categories and how they are documented.

It will be most helpful for each programmer to read over the standards before using any of the standards library; in addition, it will be helpful if the standards are followed when writing programs and submitting programs to the user's group.

11.2 PROGRAMMING STANDARDS

All subroutines will be published and distributed in symbolic assembly language or relocatable binary form.

Memory Allocations

Computer locations X7756 through X7777 octal will be used for various bootstraps, e.g., short programs for loading in the first record of a service library or service library loader from paper tapes or discs. $X=0$ for the basic memory of 4096 words. If more than one 4096-word memory module is available in the computer, X is the number of additional memory modules used.

Subroutine Call

If a subroutine requires only one parameter or argument, programmed entry will be made by first loading the desired parameter into the A register or A and B registers for a double or floating point entry, and then executing a jump and mark instruction to the subroutine.

Where more than two input parameters are required, the parameter will be entered into the program following the jump to the subroutine. The following sequence of instructions will be used.

Location	Instruction	Remarks
P	Jump and mark	Jump to subroutine.
P + 2	Parameter	Parameters or parameter locations for subroutine.
P + 3	Parameter	Parameters or parameter locations for subroutine.
P + 4	Parameter	Parameters or parameter locations for subroutine.
P + n	Parameter	Parameters or parameter locations for subroutine.
P + n + 1	Address of error routine	To execute error action.
P + n + 2	Normal return	Continuation of program.

11.3 PROGRAM DESCRIPTION

The published material for each routine constitutes a distinct package, separated materially from all other routines. (This is done to facilitate revisions and republication of the material for one routine without the necessity of republishing all others.) The published material for each routine is as follows:

a. Identification

Title
Identification
Category
Programmer
Date

b. Purpose

c. Use

Calling sequence or operational
procedure
Arguments or parameters

Space required (decimal)
Temporary storage requirements
(decimal)
Alarms or printouts
Error returns or error codes
Error stops
Input and output devices
Input and output formats
Sense switch settings
Timing
Accuracy
Limitations
Cautions to users
Equipment configuration
References

d. Method or Algorithm

e. Flow Charts

If any of the previous items are not applicable in the routine, the words "not applicable" are inserted.

Identification

Each program is identified by a category designator consisting of a classification code, program identification, and title.

The classification code consists of a letter indicating the primary class, followed by a digit indicating the subclass. The primary class is chosen from the following expandable list:

- a. Programmed arithmetic
 - 1. Fixed-point single precision
 - 2. Fixed-point double precision
 - 3. Floating-point single precision
- b. Elementary functions
 - 1. Trigonometric
 - 2. Exponential and logarithmic
 - 3. Hyperbolic
 - 4. Roots
- c. Input
 - 1. Binary
 - 2. Octal
 - 3. Alphanumeric
- d. Output
 - 1. Binary
 - 2. Octal
 - 3. Alphanumeric
- e. Executive Routines
 - 1. Assembly
 - 2. Compiling
- f. Debugging Routines
 - 1. Tracing
 - 2. Dump
 - 3. Search
 - 4. Breakpoint
- g. Diagnostic programs
- h. Service programs
 - 1. Program load and dump
 - 2. Check sum verify
- i. All others

11.4 PROGRAMMED ARITHMETIC

IDENTIFICATION

Title: Fixed single-precision integer binary-to-decimal conversion.

Identification: XBTD.

Category: A1.

Programmer: J. H. Hathwell

Date: October, 1965.

PURPOSE

XBTD converts the absolute value of the integer in the A register, modulo 10,000, to a four digit decimal coded integer in the B register. The input is retained in the A register and the X register is unchanged. The output range is 0 through 9999 inclusive.

USE

Calling sequence: Call XBTD.

Arguments or parameters: The binary argument is in the A register before and after execution.

Space required: Twenty-seven words.

Temporary storage requirements: Four words.

Alarms or printouts: None.

Error returns or error codes: None.

Error stops: None.

Input and output devices: Not applicable.

Input and output formats: Not applicable.

Sense switch settings: Not applicable.

Timing: Maximum, 138 cycles.
Average, 137 cycles.
Minimum, 136 cycles.

Accuracy: Exact.

Cautions to user: -2^{15} causes overflow and a meaningless result. Also note that result is unsigned and modulo 10000.

Equipment configuration: Not applicable.

References: Not applicable.

METHOD

Successive division of binary integer by 10_{10} with concatenation of remainders.

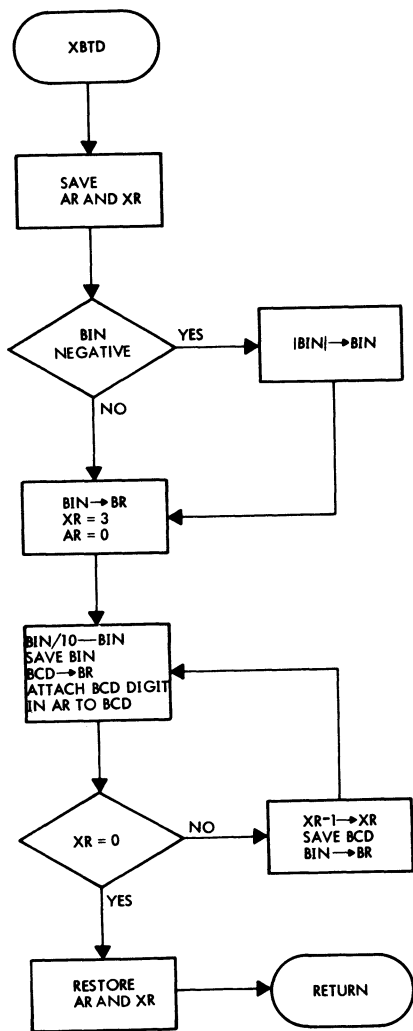


Figure 11-1. Binary-to-Decimal Conversion (Modulo 10000 decimal)

IDENTIFICATION

Title: Fixed single-precision integer decimal-to-binary conversion.

Identification: XDTB.

Category: A1.

Programmer: J.H. Hathwell.

Date: October, 1965.

PURPOSE

XDTB converts the four-digit binary-coded decimal integer in the A register to a binary integer in the B register. The input is retained in the A register with the X register unchanged. The input range is +0 through +9999 inclusive.

USE

Calling sequence: Call XDTB.

Arguments or parameters: The decimal argument is in the A register before and after execution.

Space required: Twenty-four words.

Temporary storage requirements: Four words.

Alarms or printouts: None.

Error returns or error codes: None.

Error stops: None.

Input and output devices: Not applicable.

Input and output format: Not applicable.

Sense switch settings: Not applicable.

Timing: 113 cycles.

Accuracy: Exact.

Cautions to users: Input is not checked for legal bcd codes, but is evaluated as:

$$D_3 * 10^3 + D_2 * 10^2 + D_1 * 10^1 + D_0 * 10^0$$

where D is a four-bit binary number.

Equipment configuration: Minimum configuration.

References: Not applicable.

METHOD

Successive multiplication of digits by powers of 10 with accumulation.

$$B = ((10D_3 + D_2) 10 + D_1) 10 + D_0.$$

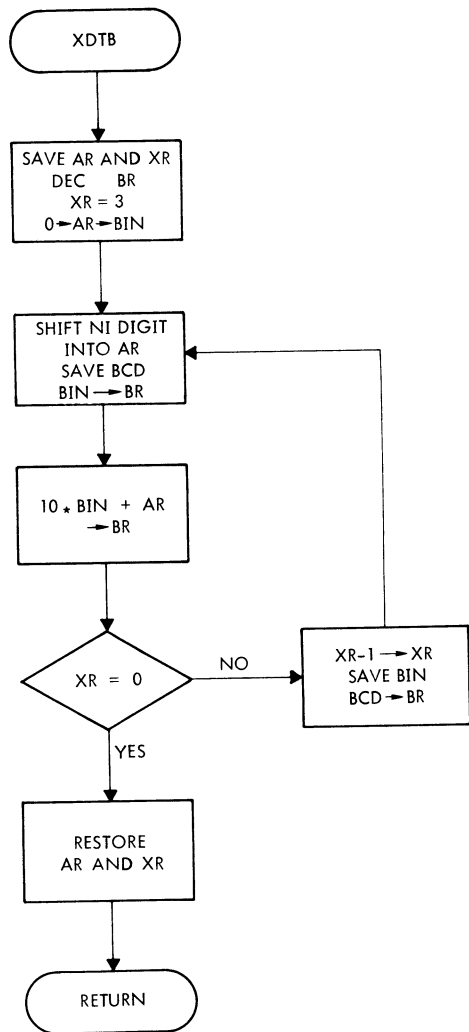


Figure 11-2. Decimal-to-Binary Conversion

IDENTIFICATION

Title: Fixed-point single-precision multiply.

Identification: XMUL.

Category: A1.

Programmer: J.H. Hathwell.

Date: October, 1965.

PURPOSE

XMUL provides the software version of the (optional) hardware multiply instruction.

USAGE

Calling sequence: LDB Multiplier
LDA Constant
CALL XMUL
PZE Address of multiplicand
Normal return.

Arguments or parameters: On entry, A = constant to be added to product. B = multiplier.

On exit, A, B = double-precision product, X is unchanged.

Space required: 38 words (046₈).

Temporary storage required: 2 words.

Alarms or printouts: None.

Error returns or codes: OV is set (1) if the product is greater than 2** (NBIT-1)-1.

Error stops: None.

Input and output devices: None.

Input and output formats or tables: None.

Sense switch settings: None.

Timing: Maximum, (B = 1) 436.75 cycles.
Average, 404.75 cycles.
Minimum, 372.75 cycles.

Accuracy: Exact.

Cautions to user: None.

Equipment configuration: Minimum.

References: Page 34

METHOD

Recursive addition of multiplicand with shifting.

IDENTIFICATION

Title: Fixed-point single-precision divide.

Identification: XDIV.

Category: A1.

Programmer: J.H. Hathwell.

Date: October, 1965.

Temporary storage required: 5 words.

Alarms or printouts: None.

Error returns or codes: OV is set (1) if the dividend is not less than the divisor.

Error stops: None.

Input and output devices: None.

PURPOSE

XDIV provides the software version of one (optional) hardware divide instruction. The true remainder and quotient are delivered to the A register and B register, respectively.

Input and output formats or tables: None.

Sense switch settings: None.

USE

Calling sequence: LDA (high dividend)
LDB (low dividend)
Call XDIV
PZE (address of divisor)
Normal return.

Arguments or parameters: On entry, A, B = double precision dividend.

On exit, A = remainder, B = quotient, X is unchanged.

Timing: Average, 200 cycles.

Accuracy: Exact.

Cautions to user: This routine produces the true quotient and remainder, i.e., $-2/1$ = quotient of -2 and remainder of zero.

Equipment configuration: Minimum.

Space required: 70 words (0106₈).

References: Page 34

IDENTIFICATION

Title: Fixed-point double-precision 2's complement.

Identification: XDCO.

Category: A1.

Programmer: J.H. Hathwell.

Date: October, 1965.

PURPOSE

XDCO takes the 2's complement of the double-precision number in the A register and B register. The X register is unchanged.

USE

Calling sequence: Call XDCO.

Arguments or parameters: The A register and the B register contain the double-precision argument before and the 2's complement after execution.

Space required: Thirteen words.

Temporary storage requirements: None.

Alarms or printouts: None.

Error returns or error codes: None.

Error stops: None.

Input and output devices: Not applicable.

Input and output formats: Double-precision numbers are stored as two successive data words. The first contains the sign and high-order 15 bits; the second contains the low-order 15 bits and is always unsigned.

Sense switch settings: Not applicable.

Timing: 9.5 cycles.

Accuracy: Exact.

Cautions to users: XDCO may set the overflow register.

Equipment configuration: Not applicable.

References: Not applicable.

METHOD

The argument is complemented and the low-order bits are tested for a carry condition.

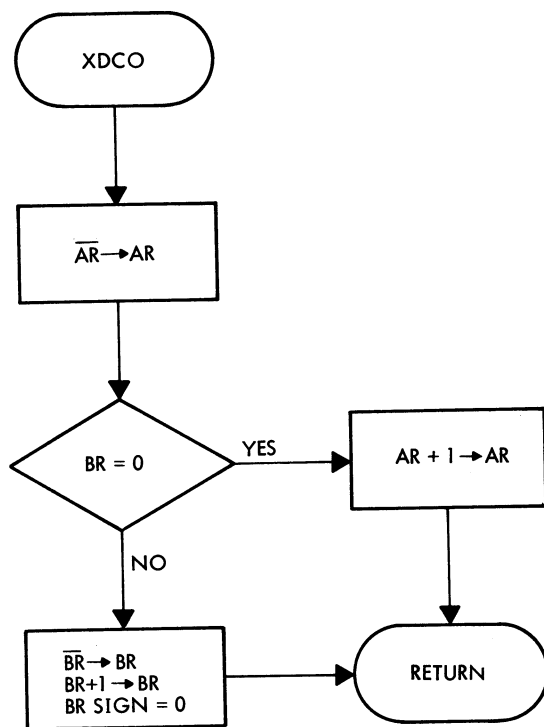


Figure 11-3. Double-precision Two's Complement

IDENTIFICATION

Title: Fixed-point double-precision add.

Identification: XDAD.

Category: A2.

Programmer: J.H. Hathwell.

Date: October, 1965.

PURPOSE

XDAD adds a double-precision number whose high-order address is in the calling sequence to the double-precision numbers in the A register and B register. The X register is unchanged.

USE

Calling sequence: Call XDAD.

PZE is the address of the double-precision augend. Normal return.

Arguments or parameters: The A register and the B register contain the double-precision addend before, and the double-precision sum after execution.

Space required: Twenty-one words.

Temporary storage requirements: Two words.

Alarms or printouts: None.

Error returns or error codes: The overflow is set if a double-precision overflow occurs.

Error stops: None.

Input and output devices: Not applicable.

Input and output formats: Double-precision numbers are stored as two successive data words. The first contains the sign and high-order 15 bits; the second contains the low-order 15 bits and is always unsigned.

Sense switch settings: Not applicable.

Timing: 30 cycles.

Accuracy: Exact.

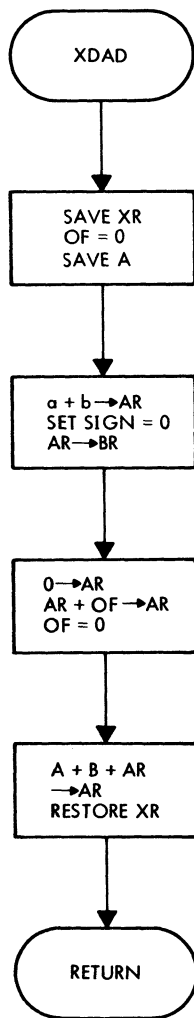
Cautions to users: The sign of the low-order words of each double precision argument must be zero to generate the proper carry. Overflow flip-flop is set on an overflow.

Equipment configuration: Minimum configuration.

References: Not applicable.

METHOD

Low-order words are added first and any carry generated is added to the high-order sum.



A = HIGH ORDER ADDEND
a = LOW ORDER ADDEND
B = HIGH ORDER AUGEND
b = LOW ORDER AUGEND

Figure 11-4. Double-precision Add

IDENTIFICATION

Title: Fixed-point double-precision subtract.

Identification: XDSU.

Category: A2

Programmer: J.H. Hathwell.

Date: October, 1965.

Alarms or printouts: None.

Error returns or error codes: The overflow is set if a double-precision overflow occurs.

Error stops: None.

Input and output devices: Not applicable.

Input and output formats: Double-precision numbers are stored as two successive data words. The first contains the sign and high-order 15 bits; the second contains the low-order 15 bits and is always unsigned.

PURPOSE

XDSU subtracts a double-precision number whose high-order address is in the calling sequence from the double-precision number in the A register and the B register. The X register is unchanged.

Sense switch settings: Not applicable.

USE

Calling sequence: Call XDSU.
PZE is the address of high-order bits of the double-minuend.
Normal return.

Timing: 32 cycles.

Accuracy: Exact.

Arguments or parameters: The A register and the B register contain the double-precision subtrahend before and the double-precision difference after execution.

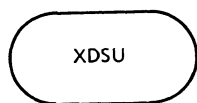
Cautions to users: The sign of the low-order words of each double-precision argument must be zero to generate the proper carry. Overflow flip-flop is set on an overflow.

Space required: Twenty-three words.

Equipment configuration: Minimum configuration.

Temporary storage requirements: Two words.

References: Not applicable.



A = HIGH ORDER SUBTRAHEND
 α = LOW ORDER SUBTRAHEND

B = HIGH ORDER MINUEND
b = LOW ORDER MINUEND

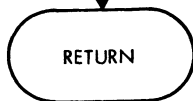
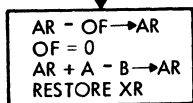
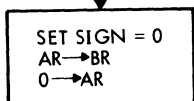
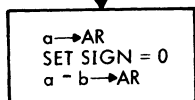
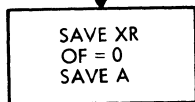


Figure 11-5. Double-precision Subtract

IDENTIFICATION

Title: Fixed-point double-precision multiply.

Identification: XDMU.

Category: A2.

Programmer: J.H. Hathwell.

Date: October, 1965.

PURPOSE

XDMU multiplies the double-precision number whose high-order address is in the calling sequence times the double-precision number in the A register and the B register. The X register is unchanged.

USE

Calling sequence: Call XDMU.

PZE is the address of the high-order bits of the multiplier. Normal return.

Arguments or parameters: The A register and the B register contain the double-precision multiplicand before and the double-precision product after execution.

Space required: Thirty-five words.

Temporary storage requirements: Three words.

Alarms or printouts: None.

Error returns or error codes: None.

Error stops: None.

Input and output devices: Not applicable.

Input and output formats: Double-precision numbers are stored as two successive data words. The first contains the sign and high-order 15 bits; the second contains the low-order 15 bits and is always unsigned.

Sense switch settings: None.

Timing: 71 cycles.

Accuracy: 2^{-30} taken as a fraction.

Caution to users: Operands should be normalized to retain precision. Overflow is reset by XDMU.

Equipment configuration: Uses hardware multiply.

References: Not applicable.

METHOD

Double-precision addition of partial products.
 $(A + a)(B + b) = AB \cdot 2^0 + AB \cdot 2^{-15} + aB \cdot 2^{-15}$

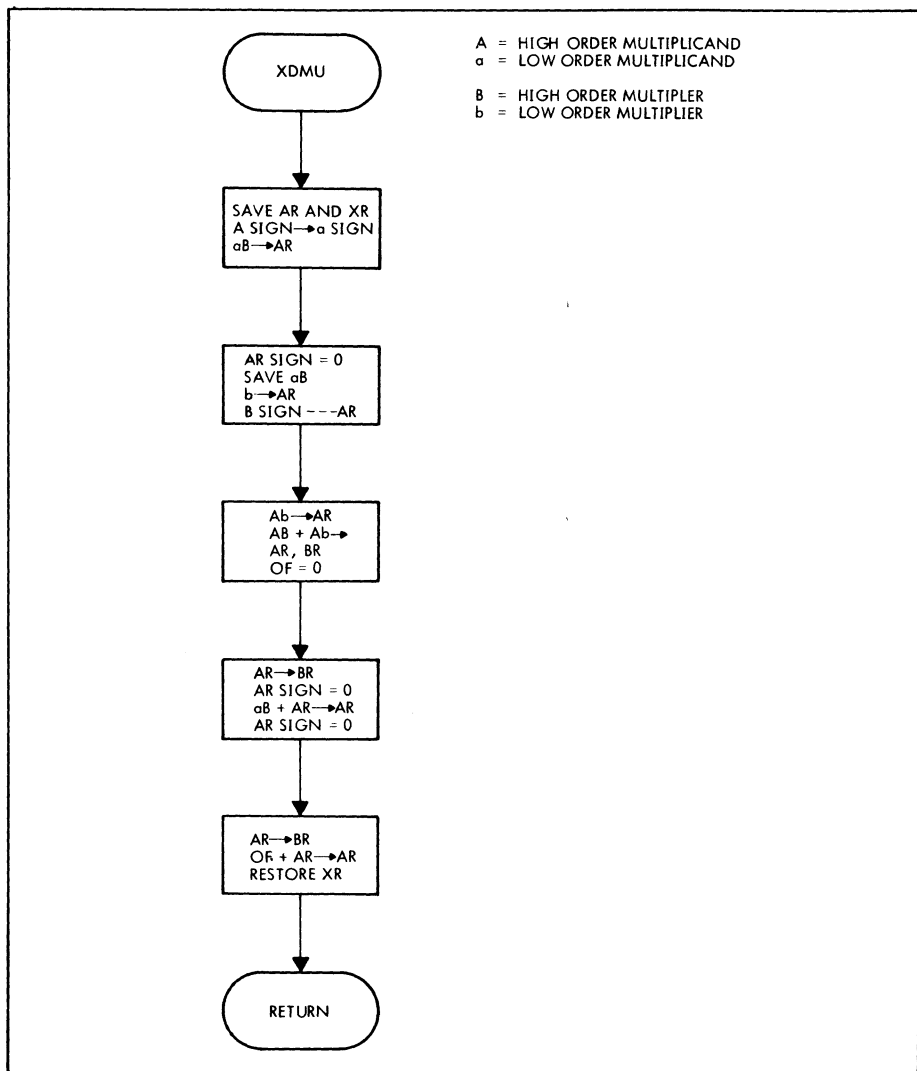


Figure 11-6. Double-precision Multiply

IDENTIFICATION

Title: Fixed-point double-precision divide.

Identification: XDDI.

Category: A2.

Programmer: J.H. Hathwell.

Date: October, 1965.

PURPOSE

XDDI divides the double-precision number in the A register and B register by the double-precision number whose high-order address is in the calling sequence. The X register is unchanged.

USE

Calling sequence: Call XDDI.
PZE is the address of high-order bits of divisor. Normal return.

Arguments or parameters: The A register and B register contain the double-precision dividend before and the double-precision quotient after execution.

Space required: Fifty words.

Temporary storage requirements: Five words.

Alarms or printouts: None.

Error returns or error codes:

Overflow is true if a divide fault occurs.

Error stops: None.

Input and output devices: Not applicable.

Input and output formats: Double-precision numbers are stored as two successive data words. The first contains the sign and high-order 15 bits; the second contains the low-order 15 bits and is always unsigned.

Sense switch settings: Not applicable.

Timing: Both areas positive, 143 cycles. Any areas negative, 172 cycles.

Accuracy: Accuracy is $\pm 2^{-29}$ taken as a fraction.

Cautions to user: Overflow is reset by XDDI. The dividend must be less than the divisor.

Equipment configuration: Hardware divide and multiply is used.

References: XDDI uses XDSU and XDCO.

METHOD

$$\frac{A + a}{B + b} \approx \frac{A + a}{B} - \frac{A \cdot b}{B^2}$$

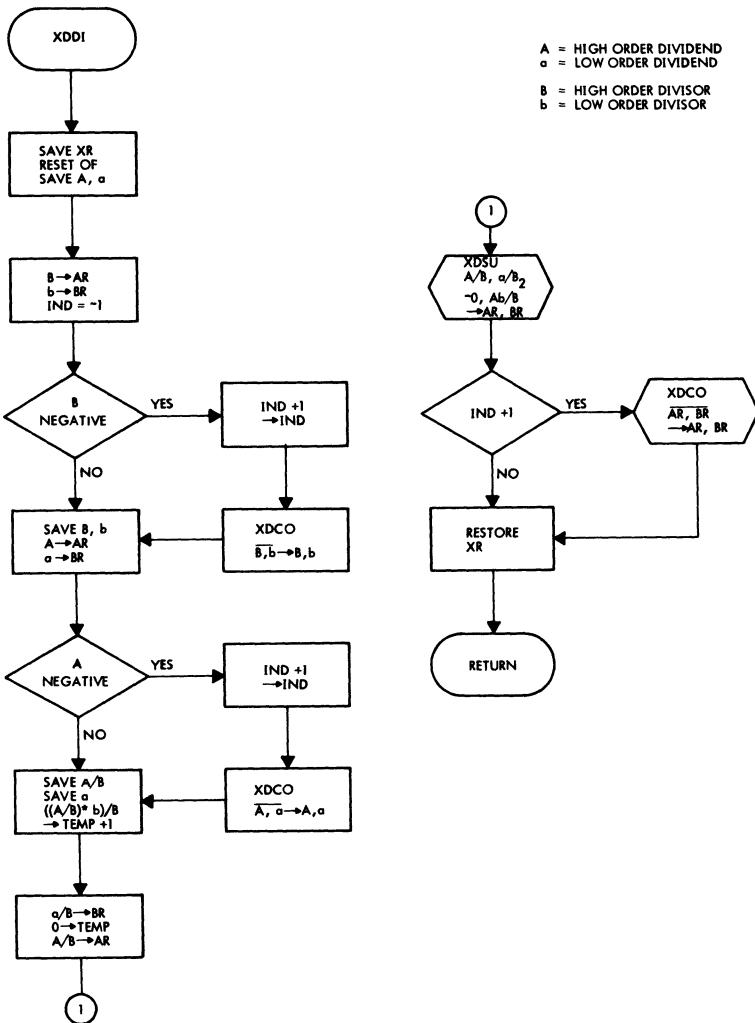


Figure 11-7. Double-precision Divide

IDENTIFICATION

Title: Absolute value, floating point (type real).

Identification: ABS.

Category: A3.

Programmer: M. McMillan.

Date: November 4, 1965.

PURPOSE:

This routine takes the absolute value of the floating-point (real) quantity in the A, B registers, returning the result to the A, B registers. The absolute value of a is defined as -a if a is negative, as a if a is not negative.

USAGE

Calling sequence: Call ABS.

Arguments or parameters: Argument is in the A, B registers.

Space required: 6 words.

Temporary storage required: Not applicable.

Alarms or printouts: Not applicable.

Error returns or codes: Not applicable.

Error stops: Not applicable.

Input and output devices: Not applicable.

Input and output formats or tables: Not applicable.

Sense switch settings: Not applicable.

Timing: Minimum, 6 cycles.
Maximum, 9 cycles.

Accuracy: No loss of information.

Cautions to user: Not applicable.

Equipment configuration: Not applicable.

References: Not applicable.

METHOD

The method is explained by the coding itself:

Label	Op Code	Variable	Comments
ABS	ENTRY JAP* CPA JMP*	ABS ABS	Return immediately if not negative. One's complement high order word if negative and return.

IDENTIFICATION

Title: Absolute value, fixed point
(type integer).

Identification: IABS.

Category: A1.

Programmer: M. McMillan.

Date: November 4, 1965.

Alarms or printouts: Not applicable.

Error returns or codes: Not applicable.

Error stops: Not applicable.

Input and output devices: Not applicable.

Input and output formats or tables: Not applicable.

PURPOSE

This routine takes the absolute value of the 16-bit signed integer in the A register and returns the result to the A register. The absolute value of a is defined as $-a$ if the a is negative and if a is non-negative.

Sense switch settings: Not applicable.

Timing: Maximum, 10 cycles.
Minimum, 7 cycles.

USAGE

Calling sequence: Call IABS.

Accuracy: No loss of information.

Cautions to users: Not applicable.

Arguments or parameters: The quantity in the A register is the argument. There are no other parameters.

Equipment configuration: Not applicable.

Space required: 7 words.

References: Not applicable.

Temporary storage required: Not applicable.

METHOD

The method is explained by the subroutine code itself:

Label	Op Code	Variable	Comments
IABS	ENTRY JAP* CPA IAR JMP*	IABS IABS	Return if argument positive or zero. If argument negative, one's complement and correct to two's complement. Return.

IDENTIFICATION

Title: Transfer of sign, fixed point
(type integer).
Identification: ISIGN.
Category: A1.
Programmer: M. McMillan.
Date: November 4, 1965.

PURPOSE

This routine applies the sign of the called (second) parameter to the quantity in the accumulator (first parameter). The parameters and result are fixed point quantities.

USAGE

Calling sequence: Call ISIGN, REF.
Arguments or parameters: The first parameter is located in the A register. The second parameter is located in core, whose address is in REF.
Space required: 27 words, including two local working cells.
Temporary storage required: Not applicable.

Alarms or printouts: Not applicable.

Error returns or codes: Not applicable.

Error stops: Not applicable.

Input and output devices: Not applicable.

Input and output formats or tables: Not applicable.

Sense switch settings: Not applicable.

Timing: Maximum, 39.75 cycles.
Minimum, 29.75 cycles.

Accuracy: No loss of information.

Cautions to user: Not applicable.

Equipment configuration: Not applicable.

References: Not applicable.

METHOD

The method is illustrated by the flowchart.
Uses \$SE.

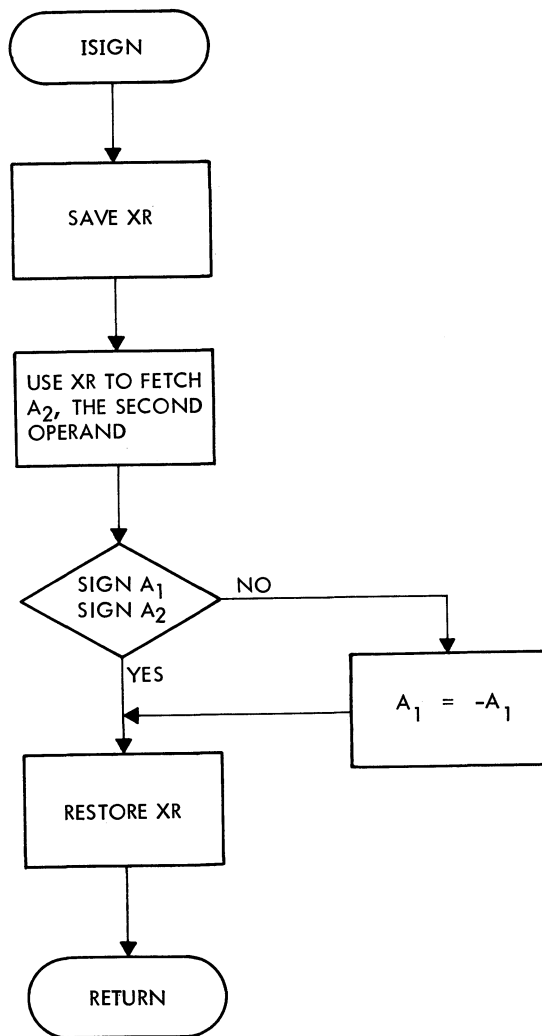


Figure 11-8. Copy Sign (Fixed Point)

IDENTIFICATION

Title: Copy sign.
Identification: SIGN.
Category: A3.
Programmer: M. C. Advani.
Date: August 31, 1966.

PURPOSE

To set sign of floating point number equal to that of argument.

USAGE

Calling sequence: Call SIGN, REF.
Arguments or parameters: Floating point number in A, B registers. REF is address of argument.
Space required: 17 words.
Temporary storage required: 2 words.

Alarms or printouts: Not applicable.

Error returns or codes: Not applicable.

Error stops: Not applicable.

Input and output devices: Not applicable.

Input and output formats or tables: Floating point format.

Sense switch settings: Not applicable.

Timing: Average, 67.5 cycles.

Accuracy: Exact.

Cautions to user: Not applicable.

Equipment configuration: Not applicable.

References: Not applicable.

METHOD

Sets sign equal to that of argument. Output in A, B registers. Uses \$SE.

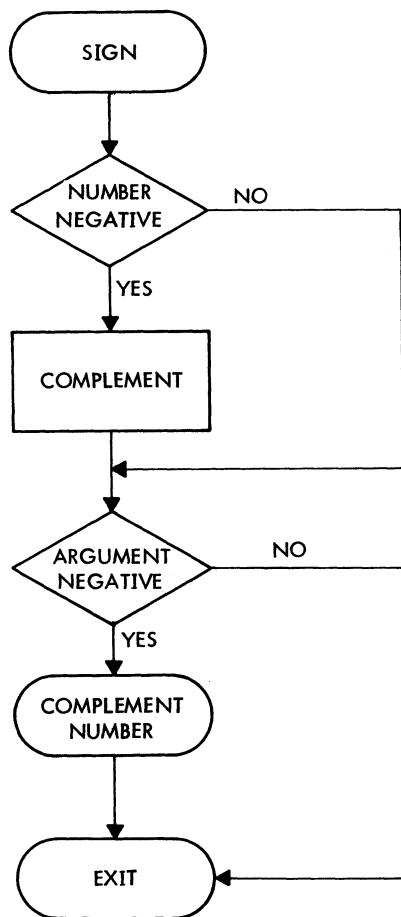


Figure 11-9. Copy Sign (Floating Point)

IDENTIFICATION

Title: Separate mantissa.

Identification: \$FMS, \$FSM.

Category: A3.

Programmer: M.C. Advani.

Date: August 31, 1966.

PURPOSE

To separate a positive floating point number into characteristic and mantissa.

USAGE

Calling sequence: Call \$FMS or \$FSM.

Arguments or parameters: A and B registers contain floating point number.

Space required: 9 words.

Temporary storage required: 1 word.

Alarms or printouts: Not applicable.

Error returns or codes: Not applicable.

Error stops: Not applicable.

Input and output devices: Not applicable.

Input and output format or tables: Floating point, input A, B contain fixed point mantissa. X contains characteristic at B8 on exit.

Sense switch settings: Not applicable.

Timing: Average, 13 cycles.

Accuracy: Exact.

Cautions to user: Not applicable.

Equipment configuration: Not applicable.

References: Not applicable.

METHOD

Output in A, B (mantissa) and X (characteristic) registers.

IDENTIFICATION

Title: Floating point number to integer.

Identification: \$HS.

Category: A3.

Programmer: M.C. Advani.

Date: August 31, 1966.

PURPOSE

To convert a floating point number to an integer.

USAGE

Calling sequence: Call \$HS, STORE.

Arguments or parameters: Number in A, B registers. STORE is address of memory where the result is to be saved.

Space required: 55 words.

Temporary storage required: 1 word.

Alarms or printouts:

Not applicable.

Error returns or codes:

If number greater than $2 * 15$ or less than 1, is exits with A, B registers set to zero.

Error stops:

Not applicable.

Input and output devices:

Not applicable.

Input and output formats or tables:

Floating point input. Fixed point integer output.

Sense switch settings:

Not applicable.

Timing:

Average, 89.5 cycles.

Accuracy:

15 bits.

Cautions to user:

Not applicable.

Equipment configuration:

Not applicable.

References:

Section 12.

METHOD

Uses \$SE.

IDENTIFICATION

Title: Normalize.
Identification: \$NML.
Category: A3.
Programmer: M.C. Advani.
Date: August 31, 1966.

PURPOSE

To normalize a double precision number.

USAGE

Calling sequence: Call \$NML.
Arguments or parameters: Number in A, B registers.
Space required: 39 words.
Temporary storage required: 2 words.
Alarms or printouts: Not applicable.

Error returns or codes: Not applicable.

Error stops: Not applicable.

Input and output devices: Not applicable.

Input and output formats or tables: Fixed point format.

Sense switch settings: Not applicable.

Timing: Average, 101 cycles.

Accuracy: 22 bits.

Cautions to user: Not applicable.

Equipment configuration: Not applicable.

References: Section 12.

METHOD

Shifts to sign and tests for sign set. Uses XDCO. Output in A, B registers. Flag for sign in X register.

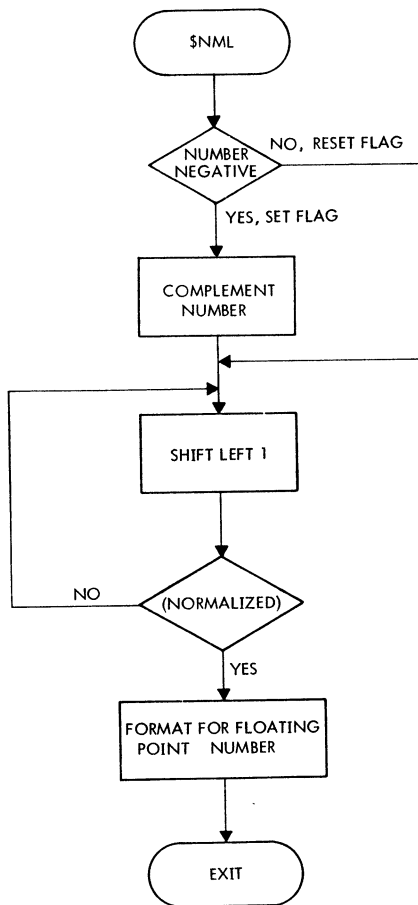


Figure 11-10. Normalize

IDENTIFICATION

Title: Floating add.
Identification: \$QK.
Category: A3.
Programmer: M.C. Advani.
Date: August 31, 1966.

PURPOSE

To add 2 floating point numbers.

USAGE

Calling sequence: Call \$QK, REF.
Arguments or parameters: A, B registers contain first argument. REF - address of second argument. Result in A, B registers.
Space required: 140 words.
Temporary storage requirements: 9 words.
Alarms or printouts: Not applicable.

Error returns or codes: Not applicable.

Error stops: Not applicable.

Input and output devices: Not applicable.

Input and output formats or tables: See floating point format.

Sense switch settings: Not applicable.

Timing: Average, 224 cycles.

Accuracy: 22 bits.

Cautions to user: Not applicable.

Equipment configuration: Not applicable.

References: Section 12.

METHOD

Algebraically adds two numbers.

\$QK and \$QL use common logic \$FAS. \$FAS determines if it is an arithmetic addition or subtraction and proceeds accordingly. \$FAS has a special entry linkage and is used solely by \$QK and \$QL.

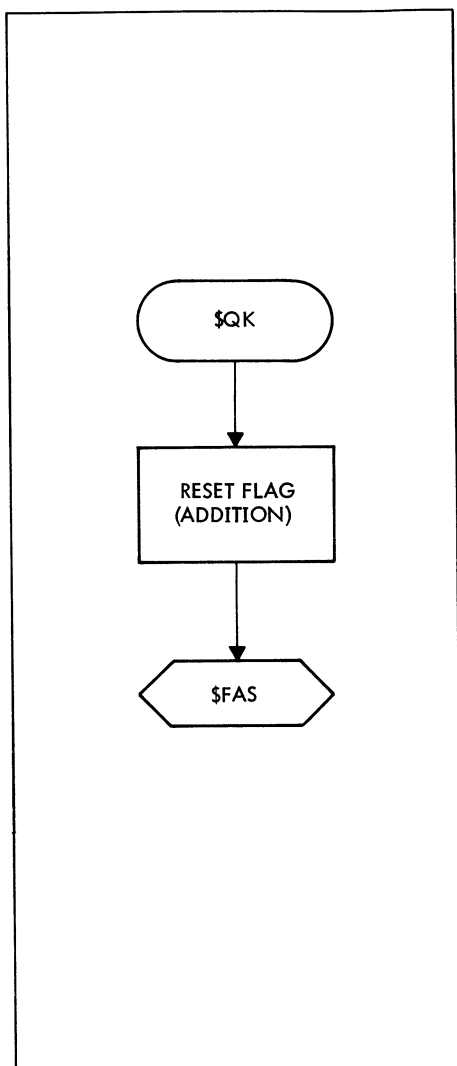


Figure 11-11. Floating Add

IDENTIFICATION

Title: Floating subtract.
Identification: \$QL.
Category: A3.
Programmer: M.C. Advani.
Date: August 31, 1966.

Alarms or printouts: Not applicable.
Error returns or codes: Not applicable.
Error stops: Not applicable.
Input and output devices: Not applicable.

PURPOSE

To compute difference of two floating point numbers.

Input and output formats or tables: See floating point format.

USAGE

Calling sequence: Cal \$QL, REF.
Arguments or parameters: Minuend in A, B, registers. REF is address of first word of subtrahend.
Space required: 4 words.
Temporary storage required: Not applicable.

Sense switch settings: Not applicable.
Timing: Average, 223 cycles.
Accuracy: 22 bits.
Cautions to user: Not applicable.
Equipment configuration: Not applicable.
References: Section 12.

METHOD
Uses \$QK.

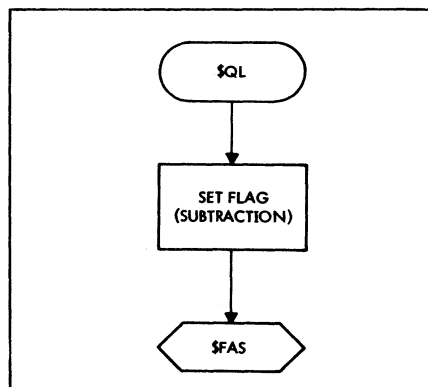


Figure 11-12. Floating Subtract

IDENTIFICATION

Title: Floating add or subtract.

Identification: \$FAS.

Category: A3.

Programmer: M.C. Advani.

Date: August 31, 1966.

PURPOSE

To provide common logic for \$QK, SQL. It has a special linkage for use by \$QK for \$QL.

USAGE

Calling sequence: Not for general use.

Arguments or parameters: Not applicable.

Space required: Included in \$QK.

Temporary storage required: Not applicable.

Alarms or printouts: Not applicable.

Error returns or codes: Not applicable.

Error stops: Not applicable.

Input and output devices: Not applicable.

Input and output formats or tables: Not applicable.

Sense switch settings: Not applicable.

Timing: Average, included with \$QK and \$QL.

Accuracy: Exact.

Cautions to user: Not for general use.

Equipment configuration: Not applicable.

References: Not applicable.

IDENTIFICATION

Title: Floating-point multiply or divide.

Identification: \$QM, \$QN.

Category: A3

Programmer: M.C. Advani.

Date: August 31, 1966.

Error returns or codes: If divisor = 0, A, B, registers set to zero and overflow on.

If result is less than $-2 * (-200g)$ or greater than $2 * (+177g)$, it returns with 0 in A, B registers and overflow on.

Error stops: Not applicable.

Input and output devices: Not applicable.

PURPOSE

To multiply 2 floating point numbers. To divide one number by another.

Input and output formats or tables: Floating point format. Output in A, B registers.

Sense switch settings: Not applicable.

USAGE

Calling sequence: Call \$QM, REF for multiply. Call \$QN, REF for divide.

Timing: Average, 237, multiply; 334, divide.

Accuracy: 22 bits multiply. 21 bits divide.

Arguments or parameters: REF is address of multiplier or divisor.

Cautions to user: Not applicable.

Space required: 126 words.

Equipment configuration: Not applicable.

Temporary storage required: 7 words.

References: Section 12.

Alarms or printouts: Not applicable.

METHOD

Separate the mantissa and use XDMU for multiply or XDDI for divide. Uses \$FMS, \$SE.

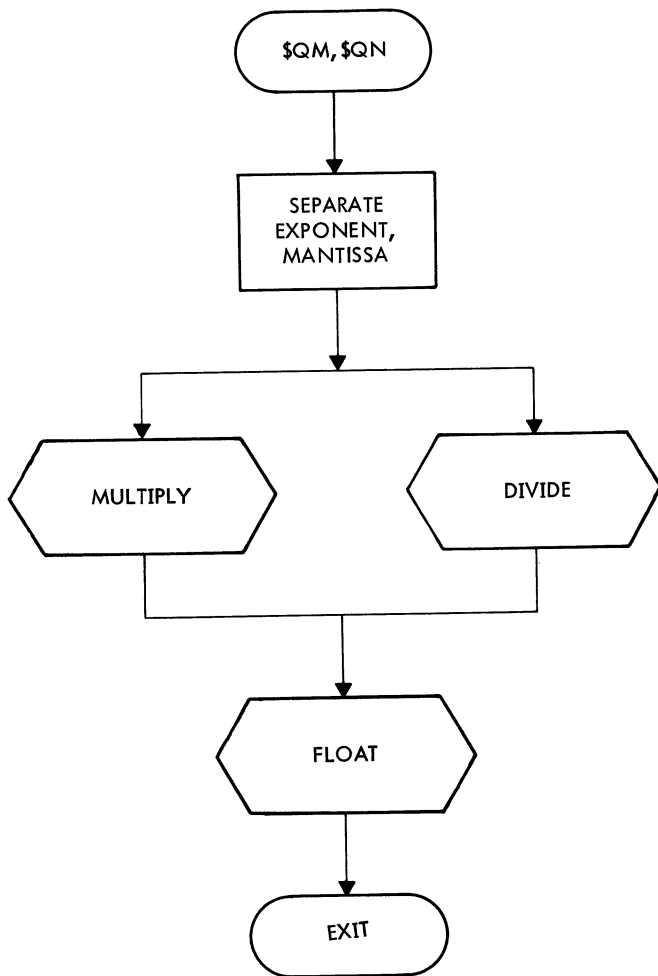


Figure 11-13. Floating-Point Multiply and Divide

IDENTIFICATION

Title: Integer number to floating-point number.

Identification: \$QS.

Category: A1, A3.

Programmer: M. C. Advani

Date: August 31, 1966.

PURPOSE

To float an integer.

USAGE

Calling sequence: Call \$QS, STORE.

Arguments or parameters: Argument in A register. STORE is address of memory where result is to be saved.

Space required: 36 words.

Temporary storage required: 3 words.

Alarms or printouts: Not applicable.

Error returns or codes: Not applicable.

Error stops: Not applicable.

Input and output devices: Not applicable.

Input and output formats or tables: Floating-point format output. Fixed-point integer input.

Sense switch settings: Not applicable.

Timing: Average, 138 cycles.

Accuracy: Exact.

Cautions to user: Not applicable.

Equipment configuration: Not applicable.

References: Section 12.

METHOD

Formats the absolute number to floating point and adjusts sign according to input. Uses \$SE.

11.5 ELEMENTARY FUNCTIONS

IDENTIFICATION

Title: Fixed single-precision logarithm.

Identification: XLOG.

Category: B2.

Programmer: M.C. McMillan.

Date: June, 1965.

PURPOSE

XLOG computes the natural logarithm of $1+X$, where the single-precision quantity X is in the A register. If

$$0 \leq X < 1,$$

the result is returned to the A register, otherwise an error exit is taken without further action. Input and output are scaled by 2^0 .

USE

Calling sequence: JMPM XLOG
JMP (error procedure)
Normal return.

Arguments or parameters: The argument X is placed in A before calling XLOG.

Space required: Eighteen words.

Temporary storage requirements: None.

Alarms or printouts: None.

Error returns or error codes: Error return if X is negative.

Error stops: None.

Input and output devices: Not applicable.

Input and output formats: Not applicable.

Sense switch settings: Not applicable.

Timing: Maximum, 203 cycles.
Average, 203 cycles.
Minimum, 9 cycles.

Accuracy: Error is less than 2^{-14} machine scale.

Cautions to users: Routine XLOG calls sub-routine POLY.

Equipment configuration: Not applicable.

References: Not applicable.

METHOD

XLOG uses a Chebychev polynomial of the fifth degree.

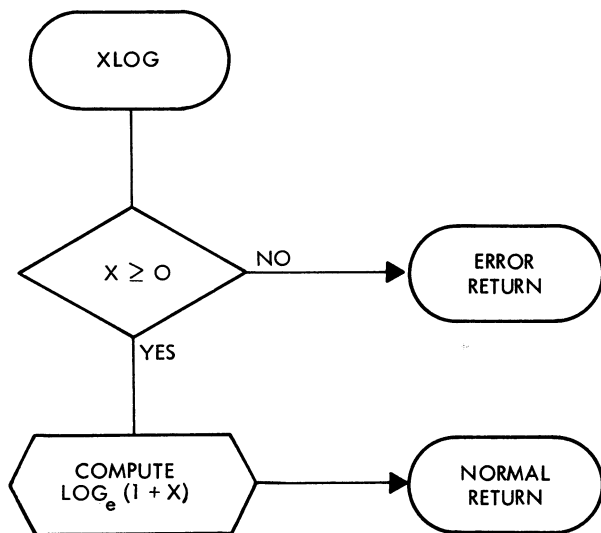


Figure 11-14. Fixed-point Natural Logarithm

IDENTIFICATION

Title: Fixed single-precision exponential, positive argument.

Identification: XEXP.

Category: B2.

Programmer: M.C. McMillan.

Date: June, 1965.

PURPOSE

XEXP computes the exponential of X, located in the A register:

$$e^X, 0 \leq X < 1$$

e^X is scaled 2^{-2} . The result is placed in the A register. (Also see PURPOSE in subroutine XEXN.)

USE

Calling sequence: JMPM XEXP
JMP (error return)
Normal return.

Arguments or parameters: The argument X is located in the A register prior to the call.

Space required: Seventeen words.

Temporary storage requirements: None.

Alarms or printouts: None.

Error returns or error codes: An error return is taken without the other action if the argument is negative.

Error stops: None.

Input and output devices: Not applicable.

Input and output formats: Not applicable.

Sense switch settings: Not applicable.

Timing: Normal, 187 cycles.
Error return, 8 cycles.

Accuracy: Error is less than 2^{-14} of machine scale.

Cautions to users: Note relative scale between input and output, and that they differ from scales relative to the routine XEXN. System subroutine XEXN is called by XEXP.

Equipment configuration: Not applicable.

Reference: Not applicable.

METHOD

The exponential is performed by means of a Chebychev polynomial of the fifth degree.

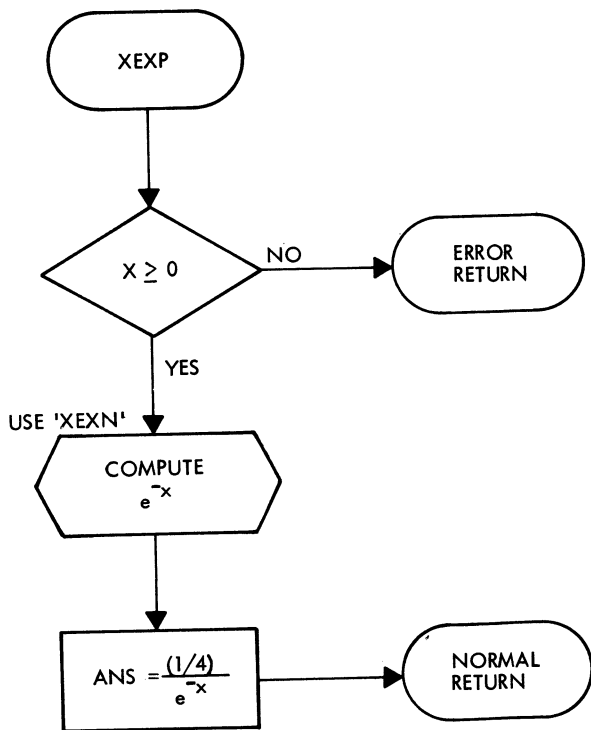


Figure 11-15. Fixed-Point Exponential

IDENTIFICATION

Title: Fixed single exponential, negative argument.

Identification: XEXN.

Category: B2.

Programmer: M.C. McMillan.

Date: June, 1965.

PURPOSE

XEXN computes the exponential of X , located in the A register:

$$e^X, -1 < X < 0$$

e^X is scaled 2^0 . The result is placed in the A register. (Also see purpose in subroutine XEXP.) The exponential was split into two subroutines, XEXP and XEXN, to increase scaling flexibility.

USE

Calling sequence: JMPM XEXN
JMP (error procedure)
Normal return.

Arguments or parameters: The argument X is located in the A register prior to the call.

Space required: Eighteen words.

Temporary storage requirements: None.

Alarms or printouts: None.

Error returns or error codes: An error return is taken without other action if the argument is negative.

Error stops: None.

Input and output devices: Not applicable.

Input and output formats: Not applicable.

Sense switch settings: Not applicable.

Timing: Normal (maximum), 159 cycles. Error return, 8 cycles.

Accuracy: Error is less than 2^{-14} of machine scale.

Cautions to users: Note that scaling conventions differ between subroutines XEXN and XEXP.

Equipment configuration: Not applicable.

References: Not applicable.

METHOD

The exponential is performed by means of a Chebychev polynomial of the fifth degree.

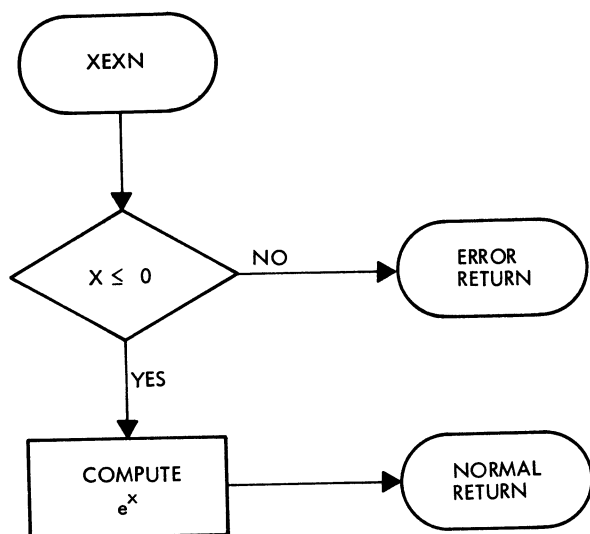


Figure 11-16. Fixed-Point Exponential

IDENTIFICATION

Title:	Fixed single-precision square root (short).
Identification:	XSQT.
Category:	B4.
Programmer:	M.C. McMillan.
Date:	October, 1965.

PURPOSE

XSQT takes the unrounded square root of the quantity in the A register if it is non-negative. The result is returned to the A register. The A register is unchanged if the input is negative.

USE

Calling sequence:	JMPM XSQT JMP (error procedure) Normal return.
Arguments or parameters:	The argument is located in the A register before execution.
Space required:	Forty-three words (forty-four if no automatic divide).
Temporary storage requirements:	Five words.
Alarms or printouts:	None.
Error returns or error codes:	Error return if argument is negative.
Error stops:	None.
Input and output devices:	Not applicable.
Input and output formats:	Not applicable.

Sense switch settings: Not applicable.

Timing: 276 cycles (hardwire divide; otherwise add 15 times the software divide time for maximum cycle time).

Accuracy: Error is less than 1.5×2^{-15} machine scale.

Cautions to users: None.

Equipment configuration: Not applicable.

References: Not applicable.

METHOD

Uses Newton-Raphson formula:

$$X_i + 1 = 1/2 X_i + \frac{A}{2X_i}, \lim X_i = \sqrt{A}$$

IN THE FORM

$$X_i + 1 = X_i + \Delta X_i$$

where

$$\Delta X_i = 1/2 \left(\frac{A}{X_i} - X_i \right).$$

If $X_0 = 1 - 2^{-15}$ (the maximum positive numeric value of a number in a 16-bit binary representation) then $\Delta X_i \leq 0$ for all steps.

If $|\Delta X_i| < 2^{-7} - 2^{-15}$ at a given step, there is no need to take another step, as would be required if testing differences of successive x-estimates.

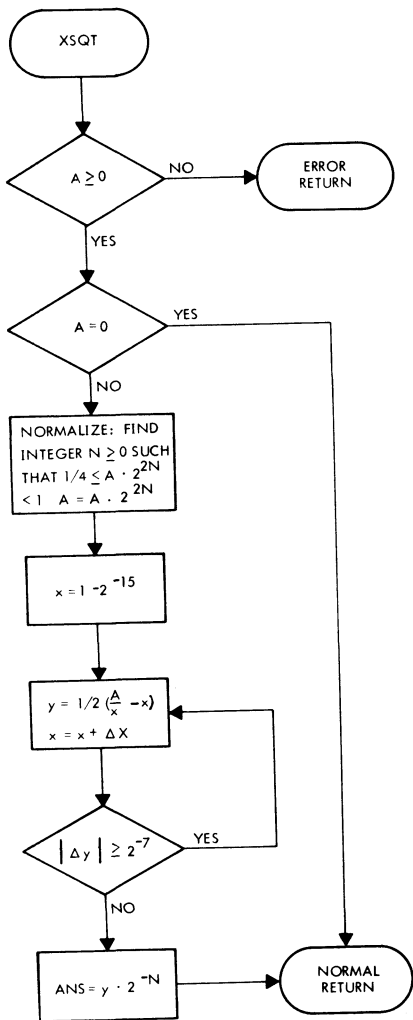


Figure 11-17. Fixed-Point Square Root

IDENTIFICATION

Title: Fixed single-precision sine.

Identification: XSIN.

Category: B1.

Programmer: M.C. McMillan.

Date: August, 1965.

Error stops: None.

Input and output devices: Not applicable.

Input and output formats: Not applicable.

Sense switch settings: Not applicable.

Timing: Maximum is typical, 175 cycles.

PURPOSE

XSIN takes the sine of the quantity X in the A register for range $-\pi \leq X \leq \pi$. The input is scaled by 2^{-2} . The output is returned to the A register, scaled 2^{-1} . X is in radians.

Accuracy: Error is less than 2^{-14} machine scale.

Cautions to users: XSIN requires subroutine POLY. No test is made for $\pi > |x| \leq 4$.

USE

Calling sequence: Call XSIN.

Arguments or parameters: The argument X in radians is in the A register.

Space required: Thirty-one words.

Temporary storage requirements: None.

Alarms or printouts: None.

Error returns or error codes: None.

Equipment configuration:

References: Not applicable.

METHOD

Uses a change of variable to y to reduce range from $(-\pi, \pi)$ to $(-\pi/2, \pi/2)$. The change of variable is $\sin x = \sin y$.

$$y = \left| X - \frac{\pi}{2} \right| - \frac{\pi}{2} \text{ if } X \geq 0$$

$$y = \left| X - \frac{\pi}{2} \right| + \frac{\pi}{2} \text{ if } X < 0$$

The Taylor sine series, truncated to five terms, is used for $\sin y$.

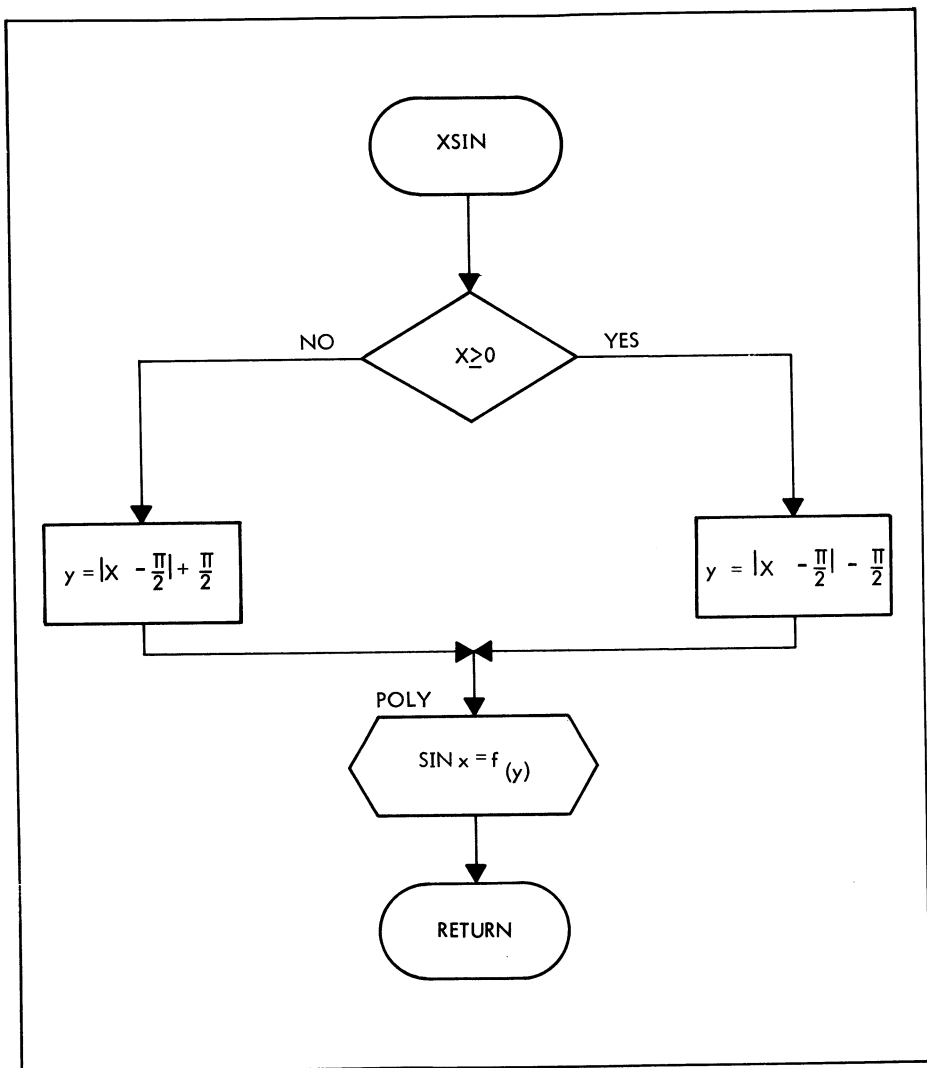


Figure 11-18. Fixed-Point Sine

IDENTIFICATION

Title: Fixed single-precision cosine.

Identification: XCOS.

Category: B1.

Programmer: M.C. McMillan.

Date: August, 1965.

PURPOSE

XCOS takes the cosine of the quantity X in A register from range $-\pi \leq X \leq \pi$. The input is scaled by 2^{-2} and the output is scaled by 2^{-1} . The output is returned to the A register.

USE

Calling sequence: Call XCOS.

Arguments or parameters: The argument X is in the A register.

Space required: Twenty words.

Temporary storage requirements: None.

Alarms or printouts: None.

Error returns or error codes: None.

Error stops: None.

Input and output devices: Not applicable.

Input and output formats: Not applicable.

Sense switch settings: Not applicable.

Timing: Maximum is typical, 172 cycles.

Accuracy: Error is less than 2^{-14} machine scale.

Cautions to users: XCOS requires subroutine POLY, no test is made for $\pi > |X| \leq 4$.

Equipment configuration:

References: Not applicable.

METHOD

Uses a change of variable to y in order to reduce the range of the variable from $(-\pi, +\pi)$ to $-\pi/2, +\pi/2$. Then $\cos x = \sin y$, where $y = \pi/2 - |X|$. The Taylor sine series, truncated to five terms, is used for $\sin y$.

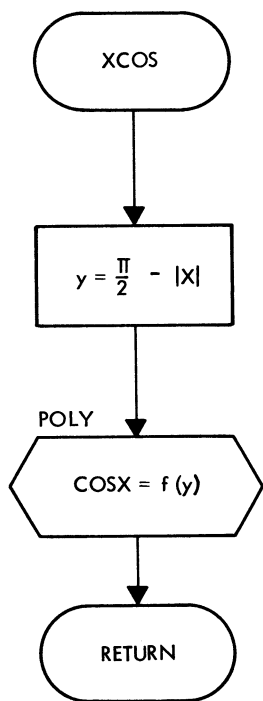


Figure 11-19. Fixed-Point Cosine

IDENTIFICATION

Title: Fixed single-precision arctangent.

Identification: XATN.

Category: B1.

Programmer: M.C. McMillan.

Date: June, 1965.

PURPOSE

XATN takes the arctangent of the quantity X in the A register, where $-1 < X < 1$. The input is scaled times 2^0 and the output is scaled times 2^0 .

USE

Calling sequence: JMPM, XATN.

Arguments or parameters: The argument X is in the A register.

Space required: Fifteen words.

Temporary storage requirements: None.

Alarms or printouts: None.

Error returns or error codes: None.

Error stops: None.

Input and output devices: Not applicable.

Input and output formats: Not applicable.

Sense switch settings: Not applicable.

Timing: Fixed, 211 cycles.

Accuracy: Error is less than 2^{-14} machine scale.

Cautions to users: XATN requires system subroutine POLY.

Equipment configuration: Not applicable.

References: Not applicable.

METHOD

XATN uses a Chebychev polynomial of seven terms. This polynomial is adequate for an 18-bit configuration.

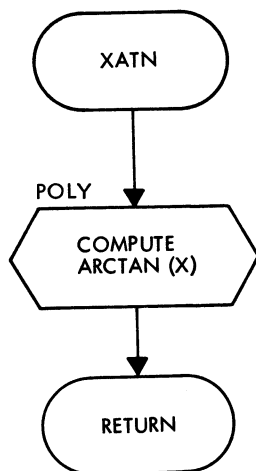


Figure 11-20. Fixed-Point Arctangent

IDENTIFICATION

Title: Single-precision polynomial.

Identification: POLY.

Category: A1. (Common Routine)

Programmer: M.C. McMillan.

Date: June, 1965.

PURPOSE

POLY is a resident utility routine intended primarily to support the fixed-point single-precision mathematical subroutines requiring the evaluation of a polynomial in one variable of any finite degree.

USE

Calling sequence: Call POLY (list of coefficients, format as below):

- a. Type code
- b. List of non-zero coefficients of degree greater than 1
- c. Zero
- d. Coefficient of degree 1
- e. Coefficient of degree 0
- f. Normal return

Arguments or parameters: The type code is either 0 or 1. Zero denotes a polynomial in all powers; one denotes a polynomial in either odd or even powers.

The list of coefficients of degree greater than one is written highest power first, and may be of any number. d) and e) coefficients must be present. Use zero to represent an absent term.

Space required: Forty-six words.

Temporary storage requirements: Three words.

Alarms or printouts: None.

Error returns or error codes: None.

Error stops: None.

Input and output devices: Not applicable.

Input and output formats: Not applicable.

Sense switch settings: Not applicable.

Timing: 40 memory cycles.
+16 memory cycles if code = 1.
+11 memory cycles if coefficient of degree 1 is not 0.
+23 memory cycles per term of degree greater than 1.

Accuracy: The accuracy attainable is close to unrounded full single-word precision.

However, accuracy obtained depends upon correct techniques of scaling and may depend on mathematical characteristics of the polynomial being evaluated.

Cautions to users: No action is taken if an additive overflow occurs during computation of the polynomial. Certain arbitrary combinations of coefficients may sharply reduce the accuracy attained. Missing interior coefficients of degrees higher than 1 must be approximated by small non-zero numbers, unless their absence is implied by type code = 1.

Equipment configuration: Not applicable.

References: Not applicable.

METHOD

The polynomial is evaluated in Horner form. For example:

$$C_4 x^4 + C_3 x^3 + C_2 x^2 + C_1 x + C_0$$

is evaluated as:

$$(((C_4 x + C_3) x + C_2) x + C_1) x + C_0$$

the parameter list taking the forms 0, C_4 , C_3 , C_2 , 0, C_1 , C_0 . The polynomial

$$C_7 x^7 + C_5 x^5 + C_3 x^3 + C_1 x$$

is evaluated as:

$$(((C_7 x^2 + C_5) x^2 + C_3) x^2 + C_1) x + 0$$

the parameter list taking the form: 1, C_7 , C_5 , C_3 , 0, C_1 , 0.

IDENTIFICATION

Title: Natural log of floating-point number.

Identification: ALOG.

Category: B2.

Programmer: M. C. Advani.

Date: August 31, 1966.

PURPOSE

To compute natural log of a floating-point number.

USAGE

Calling sequence: Call ALOG, REF.

Arguments or parameters: REF is address of argument.

Space required: 125 words.

Temporary storage required: 8 words.

Alarms or printouts: Not applicable.

Error returns or codes: Exits to \$ER if argument = 0.

Error stops: Not applicable.

Input and output devices: Not applicable.

Input and output formats or tables: Floating-point format. Output in A, B registers.

Sense switch settings: Not applicable.

Timing: Average, 907.5 cycles.

Accuracy: 21 bits.

Cautions to user: Not applicable.

Equipment configuration: Not applicable.

References: Section 12.

METHOD

$$\log A = \log_2 A * \log_e 2$$

$$\log_2 A = -1/2 + \sum_{i=0}^{i=4} C_{2i+1} Z^{2i+1}$$

$$Z = \frac{F' - \sqrt{2}}{F' + \sqrt{2}}$$

$$A = F' * 2^b$$

where

$$1 \leq F' < 2$$

$C_{2i} - 1$ are coefficients of series expansion. Uses \$ER, \$QS, \$QK, \$QM, XDMU, XDAD, \$FMS, \$NML, XDDI, XDSU, \$SE routines.

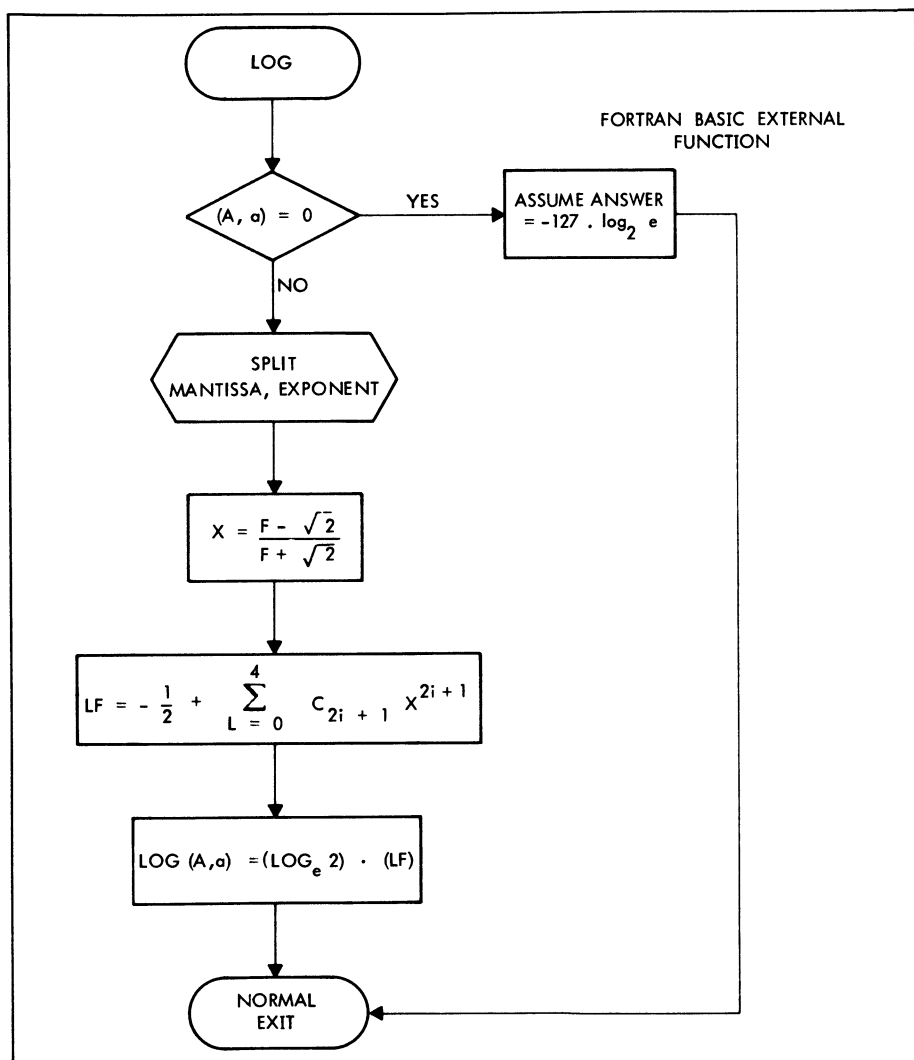


Figure 11-21. Floating-Point Natural Logarithm

IDENTIFICATION

Title: Arctangent of a floating-point number.

Identification: ATAN.

Category: B1.

Programmer: M. C. Advani.

Date: August 31, 1966.

PURPOSE

Computes arctangent of radians in floating point.

USAGE

Calling sequence: Call ATAN, REF.

Arguments or parameters: REF is address of the floating-point argument.

Space required: 184 words.

Temporary storage required: 5 words.

Alarms or printouts: Not applicable.

Error returns or codes: Not applicable.

Error stops: Not applicable.

Input and output devices: Not applicable.

Input and output formats: Floating-point format.

Sense switch settings: Not applicable.

Timing: Average, 2888 cycles.

Accuracy: 21 bits.

Cautions to user: Not applicable.

Equipment configuration: Not applicable.

References: Section 12.

METHOD

Let $N = |X|$ or $N = |X/Y|$. The arctangent of N is evaluated by dividing the total range $0 < N < 10^{75}$ into three intervals; $(10^{-5}, \tan \pi/24)$, $(\tan \pi/24, 1)$, $(1, 10^8)$. If $N < 10^{-5}$, $\arctan N = N$. If $N > 10^8$, $\arctan N = \pi/2$.

The polynomial approximation in the interval $(10^{-5}, \tan \pi/24)$ is:

$$\tan^{-1} N \approx C_1 N + C_2 N^3 + C_3 N^5.$$

Continued fraction approximations are used in the remaining intervals.

Uses \$QM, \$QL, \$QN, \$QK, \$SE routines.

$$\tan^{-1} N \approx N \cdot A_1 + (N^2 + B_2) - \frac{A_3}{(N^2 + B_3)}$$

in $(\tan \pi/24, 1)$

and

$$\tan^{-1} N \approx \pi/2 - N^{-1} D_1 -$$

$$\frac{D_2}{- (N^2 + E_2)} - \frac{D_3}{(N^2 + E_3)}$$

in $(1, 10^8)$

where

$$C_1 = 0.9999999207$$

$$C_2 = 0.3332966338$$

$$C_3 = 0.1957408066$$

$$A_1 = 0.2388229612$$

$$A_2 = 2.445205396$$

$$A_3 = 1.324747223$$

$$B_2 = 3.943529798$$

$$B_3 = 1.798249626$$

$$D_1 = 0.9999992083$$

$$D_2 = 0.3332870775$$

$$D_3 = 0.0635500089$$

$$E_2 = 0.5985998078$$

$$E_3 = 0.3953544718$$

IDENTIFICATION

Title: Cosine.
Identification: COS.
Category: B1.
Programmer: M. C. Advani.
Date: August 31, 1966.

PURPOSE

Compute cosine of angle in floating-point radians.

USAGE

Calling sequence: Call COS, REF.
Arguments or parameters: REF is address of first word of floating point number.
Space required: 24 words.
Temporary storage required: 2 words.

Alarms or printouts: Not applicable.

Error returns or codes: Not applicable.

Error stops: Not applicable.

Input and output devices: Not applicable.

Input and output formats or tables: Not applicable.

Sense switch settings: Not applicable.

Timing: Average, 1600 cycles.

Accuracy: 21 bits.

Cautions to user: Not applicable.

METHOD

Computes sine of $(\pi/2 - A)$. Uses SIN, \$QL, \$SE. Output in A, B registers.

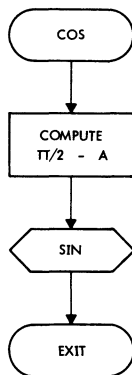


Figure 11-22. Floating-Point Cosine

IDENTIFICATION

Title: Exponential.

Identification: EXP.

Category: B2.

Programmer: M. C. Advani.

Date: August 31, 1966.

PURPOSE

To compute e^{**A} . A is floating-point number.

USAGE

Calling sequence: Call EXP, REF.

Arguments or parameters: REF is address of argument A.

Space required: 230 words.

Temporary storage required: 2 words.

Alarms or printouts: Not applicable.

Error returns or codes: Not applicable.

Error stops: Not applicable.

Input and output devices: Not applicable.

Input and output formats or tables: See floating-point format.

Sense switch settings: Not applicable.

Timing: Average, 2750 cycles.

Accuracy: 21 bits.

Cautions to user: Not applicable.

Equipment configuration: Not applicable.

References: Section 12.

METHOD

Chebyshev approximation uses XDMU, \$QK, \$QL, \$QM, \$QN, \$SE.

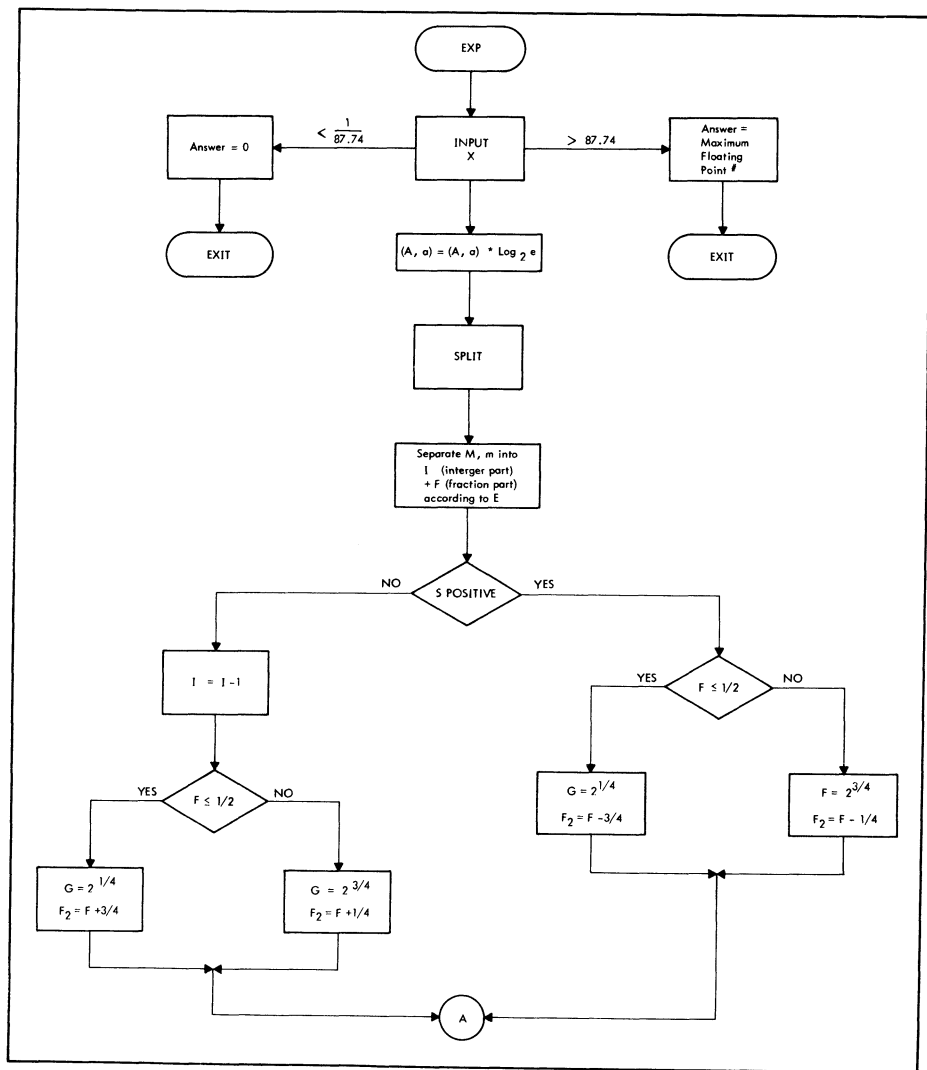


Figure 11-23. Floating-Point Exponential (Sheet 1 of 2)

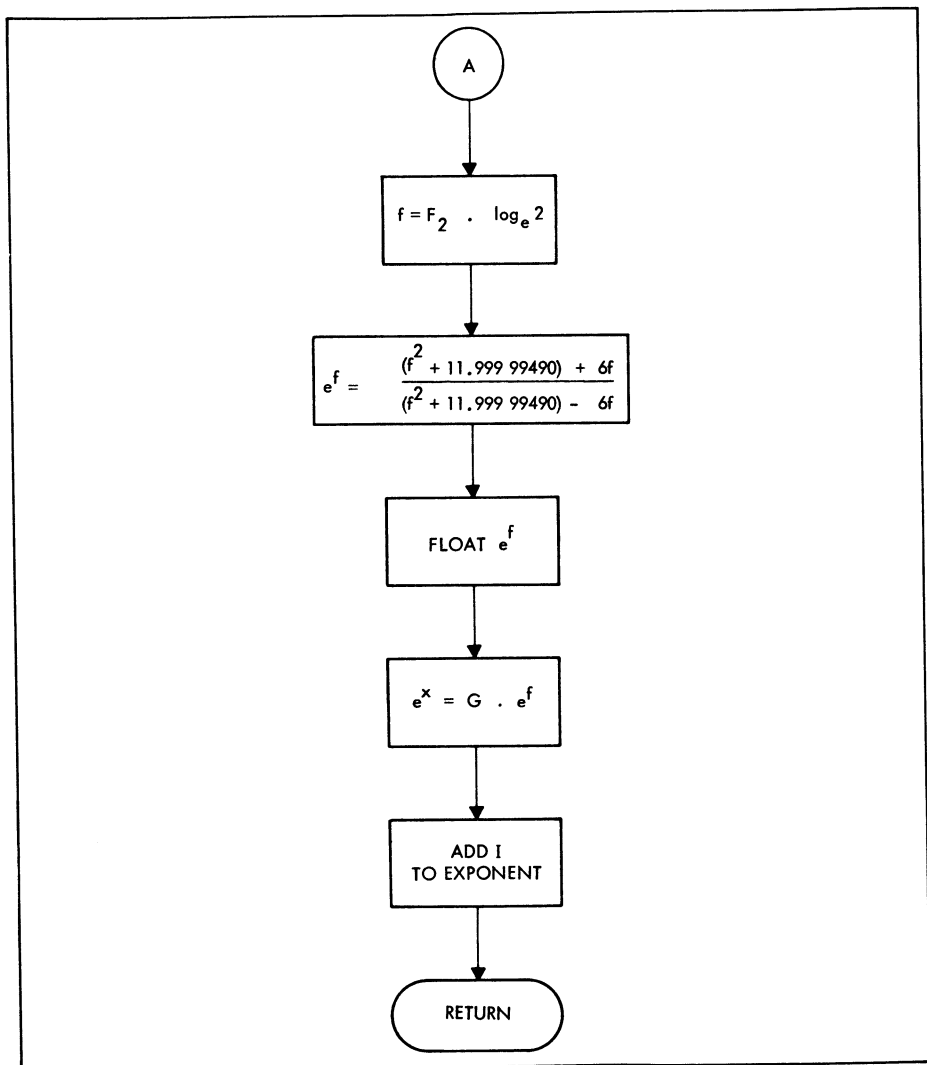


Figure 11-23. Floating-Point Exponential (Sheet 2 of 2)

IDENTIFICATION

Title: Sine.
Identification: SIN.
Category: B1.
Programmer: M. C. Advani.
Date: August 31, 1966.

PURPOSE

Compute sine of radians in floating point.

USAGE

Calling sequence: Call SIN, REF.
Arguments or parameters: Ref is address (direct or indirect) of first word of a floating-point number.
Space required: 151 words.
Temporary storage required: 6 words.
Alarms or printouts: Not applicable.

Error returns or codes: Not applicable.
Error stops: Not applicable.
Input and output devices: Not applicable.
Input and output formats or tables: See floating-point format.
Sense switch settings: Not applicable.
Timing: Average, 1305 cycles.
Accuracy: 21 bits.
Cautions to user: Not applicable.
Equipment configuration: Not applicable.
References: Section 12.

METHOD

First 5 terms of Taylor series expansion output in A, B registers. Uses \$NML, \$QM, XDMU, XDAC, \$SE, \$FMS.

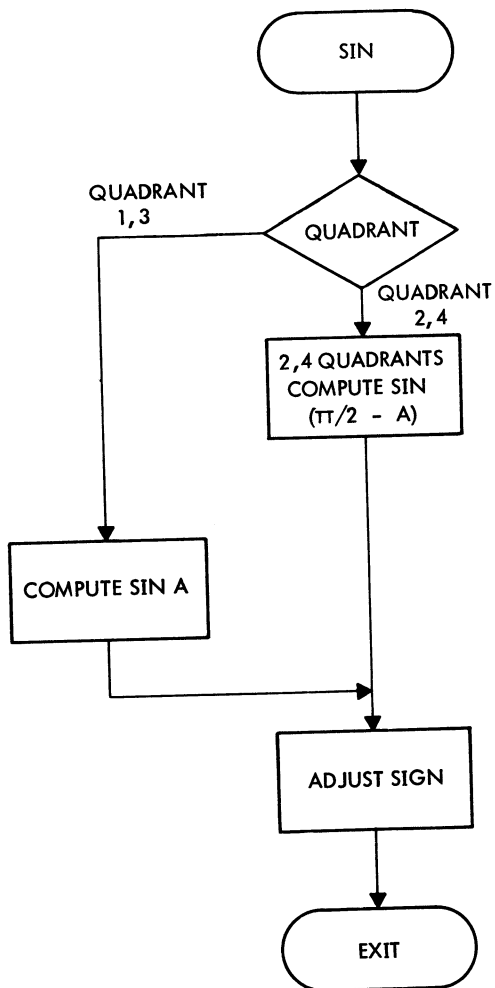


Figure 11-24. Floating-Point Sine

IDENTIFICATION

Title: Square root.

Identification: SQRT.

Category: B4.

Programmer: M. C. Advani.

Date: August 31, 1966.

PURPOSE

Computer square root of a floating point number.

USAGE

Calling sequence: Call SQRT, REF.

Arguments or parameters: REF is address of the argument.

Space required: 83 words.

Temporary storage required: 6 words.

Alarms or printouts: Not applicable.

Error returns or codes:

Exits with zero in A, B of argument negative and sets overflow flip-flop.

Error stops: Not applicable.

Input and output devices: Not applicable.

Input and output formats or tables: Floating-point format.

Sense switch settings: Not applicable.

Timing: Average, 333 cycles.

Accuracy: 21 bits.

Cautions to user: Not applicable.

Equipment configuration: Not applicable.

References: Section 12.

METHOD

Newton iteration three times. Uses \$SE, XDDI, \$FMS.

IDENTIFICATION

Title: Exponentiation of two integers.

Identification: \$HE.

Category: B2.

Programmer: M. C. Advani.

Date: August 31, 1966.

PURPOSE

To compute $I^{**}J$, I and J are integers.

USAGE

Calling sequence: Call \$HE, REF.

Arguments or parameters: I in A register. REF is address of J.

Space required: 20 words.

Temporary storage required: 2 words.

Alarms or printouts: Not applicable.

Error returns or codes: Not applicable.

Error stops: Not applicable.

Input and output devices: Not applicable.

Input and output formats or tables: Fixed-point integers.

Sense switch settings: Not applicable.

Timing: Average: 4500 cycles.

Accuracy: 15 bits.

Cautions to user: Not applicable.

Equipment configuration: Not applicable.

References: Section 12.

METHOD

Floats I and uses \$PE. Uses \$SE, \$QS, \$HS, \$PE.

IDENTIFICATION

Title: Exponentiation.

Identification: \$PE.

Category: B2.

Programmer: M. C. Advani.

Date: August 31, 1966.

PURPOSE

To compute $A^{**}I$, A is a floating-point number, I is an integer.

USAGE

Calling sequence: Call \$PE, REF.

Arguments or parameters: Argument in A, B registers.
REF is address of index I.

Space required: 21 words.

Temporary storage required: 4 words.

Alarms or printouts: Not applicable.

Error returns or codes: Not applicable.

Error stops: Not applicable.

Input and output devices: Not applicable.

Input and output formats or tables: See Floating Point.

Sense switch settings: Not applicable.

Timing: Average, 4200 cycles.

Accuracy: 20 bits.

Cautions to user: Not applicable.

Equipment configuration: Not applicable.

References: Section 12.

METHOD

Uses \$QS, \$QE, and \$SE. Floats I and goes to $A^{**}B$ (\$QE).

IDENTIFICATION

Title: Exponentiation.
Identification: \$QE.
Category: B2.
Programmer: M. C. Advani.
Date: August 31, 1966.

PURPOSE

To computer $A^{**}B$.

USAGE

Calling sequence: Call \$QE, REF.

Arguments or parameters: Argument A in A, B registers. REF is address of argument B.

Space required: 34 words.

Temporary storage required: 3 words.

Alarms or printouts: Not applicable.

Error returns or codes: Not applicable.

Error stops: Not applicable.

Input and output devices: Not applicable.

Input and output formats or tables: Floating-point format.

Sense switch settings: Not applicable.

Timing: Average, 4000 cycles.

Accuracy: 20 bits.

Cautions to user: Not applicable.

Equipment configuration: Not applicable.

References: Section 12.

METHOD

$A^{**}B$, antilog of $B \log A = e^{**} (B \log A)$.

Uses ALOG, EXP, \$SE.

11.6 UTILITY AND DEBUGGING ROUTINES

IDENTIFICATION

Title: AID II program.
Identification: AID II.
Category: F1, F2, F3, F4.
Programmer: John H. Hathwell.
Date: August 30, 1966.

PURPOSE

To provide on-line debugging.

USE

Calling sequence: Run at X6000, where X = 0 for 4K, X = 1 for 8K, etc.

Arguments or parameters: None.

Space required: 637 octal.

Temporary storage required: None.

Alarms or printouts: Insert I.

Error returns or codes: None.

Error stops: None.

Input and output devices: Model 33/35 A or B teletypes.

Input and output formats or tables: None.

Subroutines required: Self-contained program.

Sense switch settings: None.

Timing: Maximum, not applicable.
Average, not applicable.
Minimum, not applicable.

Accuracy: Not applicable.

Usage

1. Load AID II via the binary loader.
2. To enter set instruction counter = 0X6000, where X equals zero for 4K memory, X equals one for 8K memory, etc., and press RUN.
3. Three pseudo registers A, B and X are used. These registers are loaded by teletype control or trap return. The corresponding machine registers are loaded with the pseudo register values before any GOTO or trap command is executed.
4. Commands consist of a command letter (mnemonic) followed by a string of octal parameters, separated by commas and terminated by a period. In the description that follows @ indicates a carriage return/line feed type-out, upper case letters are command mnemonics (A), lower case letters are octal parameters (a), letters enclosed in parentheses denote the contents of the designated location. Underlined symbols denote AID II type-outs, all others are operator entries. A parameter preceded by a minus (-) indicates a negative parameter.
5. AID II Commands:

A (A) . @
(Display value of pseudo A.)

B (B) . @

(Display value of pseudo B.)

X (X) . @

(Display value of pseudo X.)

A (A) a . @

(Change value of pseudo A to a.)

B (B) a . @

(Change value of pseudo B to a.)

X (X) a . @

(Change value of pseudo X to a.)

G a .

(Preset A to (A), B to (B), X to (X) and go to location a.)

T a , b . @ a (a) (A) (B) (X) @

(Preset registers and go to location b. If and when location a is reached, save and type location a, the contents of a, and the current values of the registers. (Trap to a from b.)

T a , . @ a (a) (A) (B) (X) @

(Continue trap from last breakpoint location to new breakpoint location a. (Present and save registers as before.))

1 a , b , c , . @

(Initialize locations a through b (set to c).)

S a , b , c , . @ L (L) @

(Search locations a through b for words equal to c. Type out the location (L) and contents of each word thus found.)

S a , b , c , d . @ L (L) @

(Search locations a through b for words equal to c. Parameter d is used as a mask (comparison is made only for those bit positions in memory which have ones in the corresponding bits of the mask).)

S a , b , 0 , 0 . @

(Print the contents locations of a through b for zero with a zero mask, no bits selected.)

a (a) @

a + 1 (a + 1) @

a + 2 (a + 2) @

o

o

o

b - 1 (b - 1) @

b (b) @

C a . @

(Change/display memory from location a.)

a (a) , @

(Display next location (a + 1).)

a + 1 (a + 1) b , @

(Change a + 1 to value b and display next location.)

WARNING

An incomplete trap (no return to AID II) will leave the object program with the instruction at the breakpoint location changed to a return jump to the AID II trap return. These locations should be restored to their original values before further use of the trap function to ensure proper results.

IDENTIFICATION

Title: Binary load dump program.
Identification: BLD.
Category: H1, H2.
Programmer: John H. Hathwell.
Date: August 30, 1960.

PURPOSE

To load and dump programs in standard binary format.

USE

Calling sequence: Call dump (X7434) with A = 1st address, B = last address and X = execution address (if X < 0, then no execution address). Call load (X7630) with A < 0 to verify tape, A = 0 to load and return, A > 0 to load and execute.

Arguments or parameters: Subroutine entries shown above. Manual entry set A, B, and X and run at X7400 to punch bootstrap, X7404 to punch programs, X7600 to load programs.

Space required: 377 octal.

Temporary storage required: None.

Alarms or printouts: None.

Error returns or codes:

Punch: None.

Load: A = load mode, B=0 if good load or B=-1 if check sum error, X=last block address if check sum error or execution address if good load.

Error stops:

Check sum error:
IC = X7600, B = -1.

Equipment configuration:

Minimum configuration of 4096 words and teletype.

Usage

1. Initialize the paper tape reader and/or set the teletype on-line.
2. Place the program tape in the reader and place the reader control lever in the RUN position.
3. Set the A register to the load mode:

 < 0 to verify the program tape.

 = 0 to load the program tape and halt.

 > 0 to load the program tape and execute the program.
4. Set the instruction counter = X7600, press SYSTEM RESET and RUN.
5. A successful load is indicated by a halt at X7600 with the A register set to the load mode, the B register set to zero and the X register set to the execution address.

6. A checksum or format error causes a halt at X7600 with the B register set to -1 (177777) and the X register set to the load address of the last record read.
7. To restart, position the program-tape at the previous record mark and press RUN.

Procedure to Punch Program Tapes

1. Initialize the paper tape punch and/or set the teletype on-line.
2. Set the A register to the address of the first word to be punched. Set the B register to the address of the last word to be punched. Set the X register to the address of the first instruction to be executed (at load time).
3. Set the instruction counter = X7404, press SYSTEM RESET and RUN.
4. The specified memory locations will be punched and the computer will halt at X7404 with the original parameters in registers.
5. To punch noncontiguous memory areas, set the X register to -1 (177777) for all but the last area to be punched.

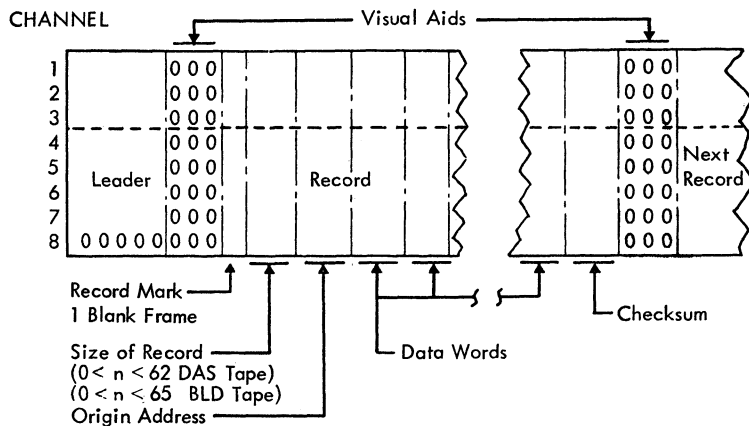
Procedure to Punch the Bootstrap Loader

1. Initialize the paper tape punch and/or set the teletype on-line.
2. Set the instruction counter = X7400, press SYSTEM RESET and RUN.
3. The loader bootstrap will be punched and the computer will halt at X7404.
4. The binary punch routine is punched following the bootstrap by setting A register = X7400, B register = X7600, X register = 00000 and press RUN.

Input and output devices: All standard peripheral devices.

Input and output formats or tables: Each word is punched three frames per word, six bits per frame, high order first. Channel 8 is not punched between visual aids. Channel 7 is the logical complement of channel 6. The checksum word is the Exclusive OR of all preceding data words, record size and origin address.

TAPE FORMAT



A record size of zero signals the end of a tape.

Timing:

Function of peripheral devices.

Sense switch settings.

None.

Accuracy:

Not applicable.

Cautions to user:

None.

IDENTIFICATION

Title: Source tape correction program.

Identification: COR.

Category: H.

Programmer: J. L. Atwood.

Date: August, 1965.

PURPOSE

This program provides an easy means by which source program statements can be added to or deleted from a paper tape, and by which superfluous non-typing characters can be eradicated.

USE

Operational procedures:

- a. Insert the source paper tape in the model 33 teletype reader and prepare the reader and punch.
- b. RUN with the instruction counter set to symbolic location SENT + 1.
- c. After the COR program has been loaded into the DATA 620/i memory, place the source paper to be corrected in the ASR-33 or ASR-35 teletype paper-tape reader. Set the DATA 620/i instruction counter display to the COR symbolic location SENT + 1. Before pressing the RUN pushbutton, set sense switches 1 and 2 to the desired condition.

Sense switch settings have the following meaning:

SENSE Switch 1:

Off, the next code line read by COR will be reproduced.

On, the next code line (source statement) read by COR will be deleted and not reproduced on the updated source tape.

SENSE Switch 2:

Off, the computer halts between code lines, allowing the DATA 620/i programmer to insert new code lines into his program. After all of the new code lines have been added, the RUN pushbutton on the DATA 620/i is pressed and the next code line for the paper tape to be updated is read into the computer.

On, the computer does not halt between code lines.

It can be seen that it is possible to delete, insert and replace code lines by using combinations of settings of SENSE switches 1 and 2. Each statement punched onto the updated paper tape is listed on the teletype, providing the programmer with a listing of his updated program.

Observe the following rules during operation of the COR system:

Each code line that is inserted into the updated program should begin with the carriage return and line feed characters.

The setting of SENSEswitch 1 should only be changed when the DATA 620/i is in the halt condition.

The setting of SENSEswitch 2 may be changed at any time during operation of the COR system.

SENSEswitches 1 and 2 should not both be on at the same time.

- d. Continue with step c until all source statements have been processed. A halt will occur at the end of the input tape regardless of SENSE switch 2 selection.

BOOTSTRAP LOADER

This is not a subroutine as such, but is a necessary program. This program is typically used when a "cold start" is required. A cold start usually occurs when the specific contents of memory is not known to the operator.

The procedure for loading the program is shown below. Use only those procedures which apply to specific system configuration.

- (1a) Turn on paper tape reader.
- (1b) Turn on model 33A teletype.
- (1c) Place model 33/35B teletype in off-line mode and press control and D, T and Q to initialize teletype.
1. Position the tape in the reader with the binary frame at the read station.
2. Set the reader control lever in the STOP or LOAD position and set the teletype on-line. For paper tape reader, no action required.

3. Enter the appropriate bootstrap load routine into memory through the console. See table of bootstrap load routines.

- a. Enable repeat.

- b. Enter either STA, STB or STX, with the address mode relative to P, into the U register (054000, 064000, 074000).

- c. Set P register to X7756.

- d. Enter a bootstrap instruction from the proper column of table 11.1 into the A, B or X register.

- e. Press STEP.

- f. Reset the A, B or X register.

- g. Repeat steps d, e and f for each bootstrap instruction.

4. Set A register = B register = zero, instruction counter = X7770, X = X7600, press SYSTEM RESET and RUN.

5. To initiate loading, set the reader control lever in the START or RUN position.

6. A successful load of the loader and punch program is indicated by a halt at X7600 with B register = zero and the reader halted.

7. Common causes for failure are:

- a. The proper bootstrap load routine was not in memory.

Table 11.1. DATA 620/i Bootstrap Load Routines

Location	Input Source			Symbolic Code
	High-Speed Reader	Model-B Teletype (620-60B, 61B)	Model-A Teletype (620-60A)	
X7756	102637	102601	102600	READ, CIB, RDR
X7757	004011*	004011*	004011*	, ASLB, NBIT-7
X7760	004041	004041	004041	, LRLB, 1
X7761	004446	004446	004446	, LLRL, 6
X7762	001020	001020	001020	, JBZ, SEL
X7763	0X7772	0X7772	0X7772	
X7764	055000	055000	055000	, STA, 0, 1
X7765	001010	001010	001010	, JAZ, LHLT + 1
X7766	0X7600	0X7600	0X7600	
X7767	005144	005144	005144	, IXR,
X7770	005101	005101	005101	ENTR, INCR, 1
X7771	100537	102601	100000	SEL, SEL, RDON
X7772	101537	101201	101100	, SEN, IBFR, READ
X7773	0X7756	0X7756	0X7756	
X7774	001000	001000	001000	, JMP, *-2
X7775	0X7772	0X7772	0X7772	

*For 18-bit computers insert 4013.

X = 0 for a 4K memory, X = 1 for an 8K memory, etc.

- b. The bootstrap was not positioned correctly.
- c. The registers were not set correctly.
- d. The teletype was not on-line

BINARY LOAD/DUMP

These programs are distributed in object form on a single tape labeled binary load/dump. The binary load program is in a special format called bootstrap format and the dump program is in standard binary format.

Essentially what happens is as follows:

1. Using a bootstrap loader (discussed in previous section) the binary loader is loaded into memory.
2. Upon completion of the load process, control is transferred to the binary loaded (recently unloaded) and;
3. It then loads the binary dump program into memory.

Procedure to Load Program Tapes

1. Initialize the paper tape reader and/or set the teletype on-line.

2. Place the program tape in the reader and place the reader control lever in the RUN position.

3. Set the A register to the load mode:

< 0 to verify the program tape.

= 0 to load the program tape and halt.

> 0 to load the program tape and execute the program.

4. Set the instruction counter = X7600, press SYSTEM RESET and RUN.

5. A successful load is indicated by a halt at X7600 with the A register set to the load mode, the B register set to zero and the X register set to the execution address.

6. A checksum of format error causes a halt at X7600 with the B register set to -1 (177777) and the X register set to the load address of the last record read.

7. To restart, position the program tape at the previous record mark and press RUN.

SECTION 12 FORTRAN

12.1 BASIC FORTRAN CONCEPTS

12.1.1 INTRODUCTION

FORTRAN is a universal, problem oriented programming language designed to simplify the computer solution of mathematical and engineering problems. The syntactical rules for the use of the language are rigorous and require the programmer to reduce the solution characteristics of his problem to a series of precise statements. These statements are evaluated and interpreted by a system program (called the FORTRAN processor) and are translated into the execution language of the computer system.

The variations between computer systems is responsible for the development of many versions of the FORTRAN language. This condition affects the number, form and relationship of the statements acceptable to a given FORTRAN processor. It is essential, therefore, that the programmer be familiar with the language specifications for the system of intended use. DATA 620/i series FORTRAN conforms with the proposed American standards for basic FORTRAN, as published by the American Standards Association on 10 March 1965.

This section is intended for use in DATA 620/i FORTRAN programming training classes or seminars, and as a reference for experienced programmers using the DATA 620/i FORTRAN system.

12.1.2 CHARACTER SET

A FORTRAN program unit is written using the following letters, digits, and special characters:

Letters: A B C D E F G H I J K L
M N O P Q R S T U V W
X Y Z

Digits: 0 1 2 3 4 5 6 7 8 9

Special
Characters: (blank or space)
= (equals)
+ (plus)
- (minus)
* (asterisk)
/ (slash)
((left parenthesis)
) (right parenthesis)
, (comma)
. (decimal point)

With the exception of the specific uses indicated in the following sections of this manual, a blank character has no meaning, and may be used freely by the programmer to improve the readability of the FORTRAN program.

The following special characters are classified as arithmetic operators and are significant in the unambiguous statement of arithmetic expressions:

+ (addition or positive value)
- (subtraction or negative value)
* (multiplication)
/ (division)
** (exponentiation)

The special characters equals (=), open parenthesis ((), closed parenthesis ()), comma (,) and decimal point (.) have specific application in the syntactical expression of the FORTRAN statement. The following sections of this manual will qualify their use in particular statements and expressions.

In addition to the FORTRAN character set, the DATA 620/i series FORTRAN system will accept the following characters in Hollerith fields:

S, !, ", #, %, &, ', :, ;

12.1.3 LINE FORMAT

A FORTRAN program consists of a series of statements divided into physical sections called lines, that must be coded to a precise grammatical format. FORTRAN statements fall into two broad classes, executable and non-executable. Executable statements specify program action, while non-executable statements describe the use of the program, the characteristics of the operands, editing information, statement functions, or data arrangement. The statements of a FORTRAN source program are normally written on a standard FORTRAN coding form.

Figure 12-1 is a sample FORTRAN coding form. The coding form includes 80 columns of information. Columns 73 through 80 are reserved for sequencing information, and have no effect upon the generated execution program. Columns 1 through 72 contain line information in the following format:

Initial Line

The first line of each statement is called an initial line. A statement may include an initial line and continuation lines. Statements may have as many continuation lines as required subject to the following restrictions: DO statements must be wholly contained on an initial line; and the equals character (=) of a replacement statement must appear on the initial line. An initial line may contain a statement label in columns 1 through 5. In this case, column 6 must contain a zero digit, blank or space character, and columns 7 through 72 may contain all or part of a statement with the exception of the restrictions noted.

Example:

1	5	6	7	10	15	20	25	30	35
			A =	.	5	*	C	*	D

Continuation line

Continuation lines are used when additional lines of coding are required to complete a statement originating on an initial line. There may be any number of continuation lines per statement with the exceptions previously noted for initial lines. In a continuation

line, columns 1 through 5 are ignored and should, but need not, be blank; column 6 must contain any character other than a zero digit, blank or space character; and the continued segment of the statement is contained in columns 7 through 72. Continuation lines may only follow an initial line or another continuation line.

PROGRAM	MATRIX MULTIPLICATION		PAGE NO.	OF
PROGRAMMER			IDENTIFICATION	
DATE			CLASS.	NOV 67


 version data machines
 a when mandatory

C FOR COMMENT		FORTRAN STATEMENT											
Statement Label	10	15	20	25	30	35	40	45	50	55	60	65	70
1													
2													
3													
4													
5													
6													
7													
8													
9													
10													
11													
12													
13													
14													
15													
16													
17													
18													
19													
20													
21													
22													

Figure 12-1. Sample FORTRAN Coding Form

Example:

1	5	6	7	10	15	20	25	30	35
			A =						
		1	. 5 *						
		2	C +						
		3	D						

Comments Lines

Any line with the character C in column 1 is identified as a comment line. Comments may appear anywhere in a program, except immediately before a continuation line. All

comments lines are ignored by a FORTRAN processor, except for display purposes. Comments may be contained in columns 2 through 72.

Example:

1	5	6	7	10	15	20	25	30	35
C			THIS IS A	COMMENTS	LINE				

End Line

Any line not containing the letter C in column 1 and having only the character string END in columns 7 through 72 is recognized by the processor as an end line. Each

FORTRAN program requires an end line to inform the processor that it has reached the end of that program.

Example:

1	5	6	7	10	15	20	25	30	35
			END						

Statement Label

Labels permit statements to be referenced by other portions of a program. A statement label is an integer value in the range 1 to 99999 (leading zeros or blanks are not significant for label identification). The initial line of

each statement may be given a unique label in columns 1 through 5. The same label may not be given to more than one statement in a program unit.

Example:

1	5	6	7	10	15	20	25	30	35
5,0		A,=	. 5	* C, +, D,					
6,0		A =	. 5	* C + D					
8,7,9		A,=	. 5	* C, +, D,					

12.2 DATA

12.2.1 GENERAL

Numerical quantities, constants and variables are distinguished in FORTRAN as a means of identifying the nature and characteristics of the numerical values encountered in program execution. A constant is a quantity whose value is explicitly stated. A variable is a numerical quantity referenced by name, rather than by its explicit appearance in a program statement. During the execution of a program, a variable quantity may assume many different values.

12.2.2 DATA TYPES

The DATA 620/i FORTRAN processor recognizes two types of data, integer and real. Integer data are precise representations of integral values within the range -32767 to $+32767$ ($-2^{15} + 1$ to $2^{15} - 1$). Real data are approximations of real numbers with magnitudes in the range 0.588×10^{38} to 0.588×10^{38} (approximately 2^{-127} to $2^{127} \times (1-2^{-22})$). Both integer and real data may assume positive, negative, or zero values. The value zero is considered neither positive nor negative.

12.2.3 DATA NAMES

FORTRAN data (constants, variables, arrays and array elements) are identified by names.

Symbolic Names

Symbolic names are made up of letter or digit strings consisting of 1 to 6 characters. The first character of the string must be a letter. Data identified by symbolic names are specified as being of type integer or real by the unique classification associated with the first letter of the character string. Names beginning with the letters I, J, K, L, M and N are type integer, and the names beginning with any other letters are type real.

Examples of type integer symbolic names are:

I 12A MZXF N5

Examples of type real symbolic names are:

A B2 F5M79 AAA

12.2.4 VARIABLES

Variables are data whose values are derived and defined during program execution, and are identified by symbolic names of the appropriate type, real or integer.

12.2.5 CONSTANTS

Constant data are identified explicitly by naming their actual values. Constants do not change in value during program execution, and are specified to be of type integer or real.

Integer Constants

An integer constant is identified by a non-empty string of from 1 to 5 decimal digits written without a decimal point and optionally preceded by a plus (+) or minus (-) sign character.

Examples:

-217 -32767 +00327 512

Real Constants

A real constant may consist of 1 to 7 significant digits and may be identified in any one of the following forms:

$\pm i$, $\pm i.f$, $\pm .fE\pm e$, $\pm iE\pm e$
 $\pm .f$, $\pm i.E\pm e$, $\pm i.fE\pm e$

where i , f and e are each a string of decimal digits representing an integer, fraction and exponent respectively. The plus (+) and minus (-) sign characters are optional, and the decimal point (.) and E characters are present in that form. If r represents any of the forms preceding $E\pm e$, i.e., $rE\pm e$, then the real constant is interpreted as $r \times 10^{\pm e}$.

Examples:

17. -25.620E-1 0.0 -51E1
+.42 -.479 -479E-3 .35E02

If a real constant is specified with more significant digits than the precision real data allows, truncation occurs and only the most significant digits within the range will be represented.

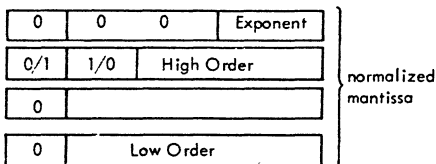
Double-Precision Constants

A double-precision constant in a source statement is specified exactly as an E-format real constant, except that the letter E is replaced by D.

Examples:

-3476.2D-4 28.D0 .578E+3

A double-precision number in memory has the following format:



12.2.6 ARRAYS

An array is an ordered set of data in 1, 2 or 3 dimensions identified by a symbolic name. An array declarator (see DIMENSION statement) defines the name and size of the array. An array name serves to identify all of the elements in the array, including data type, real or integer. An array name cannot be used without a subscript, except in input/output lists.

Array Element

An array element is one member of an array and is identified by a subscript appended to the array name.

Subscripts

A subscript follows the array name and contains 1, 2 or 3 subscript expressions enclosed in parentheses. The number of subscript expressions (except in EQUIVALENCE statements) corresponds to the specified dimensionality of the array. Two expressions within the parentheses must be separated by a comma. Subscript expressions are type integer in one of the following forms:

$$c*v \pm k \quad c*v \quad v \pm k \quad v \quad k$$

where c and k are integer constants and v is an integer variable.

Examples:

$$X(2*J-3) \quad A(I,J,K) \quad B(20) \quad C(L-2)$$

Dimensionality

Arrays are stored column-wise in ascending memory locations. Therefore, a two-dimensional array, A , with three rows and three columns would be stored internally in the computer as follows:

Location	Element
L+0 & L-1 *	A(1,1)
L+2 & L+3	A(2,1)
L+4 & L+5	A(3,1)
L+6 & L+7	A(1,2)
L+8 & L+9	A(2,2)
.	.
.	.
.	.
L+16 & L+17	A(3,3)

*In DATA 620/i FORTRAN, a storage unit for a real or integer entity is two words in length.

The position of an array element, $A(i,j)$ is derived from the following formula:

$$A_0 + (i-1 + 1 \times (j-1)) \times 2$$

where A_0 is the location of the first element in the array, i and j are the specified row and column subscript expressions, and 1 is the number of row elements defined in the array declarator for A . In the preceding example, the position of the $A(2,2)$ element would be solved in the following form:

$$L + 0 + (2-1 + 3* (2-1)) \times 2 = L + 8$$

The processor collects all constant terms in subscript expressions into the base address of the referenced array.

12.3 SPECIFICATIONS AND STATEMENTS

12.3.1 GENERAL

Specification statements organize and classify data that will be referred to by other statements in the FORTRAN program. Specification statements include:

DOUBLE PRECISION: Allows variable, array, and function names to be typed double precision.

DIMENSION: Names and declares the size of an array.

COMMON: Assigns variable and/or named arrays to common storage areas.

EQUIVALENCE: Assigns variables and names arrays to shared storage areas.

Specification statements must appear in the FORTRAN program in the order of: DOUBLE PRECISION statements, DIMENSION statements, COMMON statements and EQUIVALENCE statements.

Examples:

Valid	Invalid
DOUBLE PRECISION A	DOUBLE PRECISION A
DIMENSION D(3)	COMMON A, B, C
COMMON A, B, C	DIMENSION D(3)
EQUIVALENCE (B, D(3))	EQUIVALENCE (B, D(3))

12.3.2 DOUBLE-PRECISION STATEMENT

Form: DOUBLE PRECISION $a_1, a_2, \dots, a_n$, where a_i is a variable, array or function name.

This statement must appear before any COMMON or EQUIVALENCE statement that uses any of the names in the list. There may be any number of names in a list and any number of DOUBLE-PRECISION statements in a program. Once typed, the names remain typed throughout the program and may not be changed.

12.3.3 DIMENSION STATEMENT

Form: DIMENSION $v_1(i_1), v_2(i_2), \dots, v_n(i_n)$, where each $v(i)$, (called an array declarator) is composed of a declarator name v , (the name of the array), and a declarator subscript (i). Each (i) is an unsigned integer constant, two unsigned integer constants separated by a comma or three unsigned integer constants separated by commas. Each constant

must have a value greater than zero and less than the limit of available memory.

A DIMENSION statement specifies that the declarator names listed are arrays in the program unit. The number of dimensions and the maximum size of each dimension is specified by the declarator subscript associated with each declarator name.

More than one DIMENSION statement may appear in a program, but can only be preceded by a FUNCTION, SUBROUTINE or a previous DIMENSION statement.

An array element is referred to by the array name qualified by a subscript to identify the desired element. If the value of this subscript is out of the range specified by the array declarator, the derived computational results will be unpredictable.

Array elements are stored column-wise in computer memory from low address storage to high address storage. Therefore, one dimension arrays are stored sequentially in the order $A_1, A_2, \dots, A_n$, while two dimension arrays are stored with the first (leftmost) dimension varying most rapidly, i.e., $A_{1,1}, A_{2,1}, \dots, A_{m,1}, A_{1,2}, A_{2,2}, \dots, A_{m,n}$.

Example:

```
DIMENSION A(5), I1(3,6), C(5,10),
BIG (10,10,10)
```

This specification statement indicates that A is a real vector with 5 elements, I1 is an integer matrix of size $3 \times 6 = 18$ elements, C is a real matrix of size $5 \times 10 = 50$ elements, and BIG is a real matrix of size $10 \times 10 \times 10 = 1000$ elements.

12.3.4 COMMON STATEMENT

Form: COMMON $a_1, a_2, \dots, a_n$, where each a is a non-dummy variable or array name.

A COMMON statement specifies that the variables and/or arrays listed are to be assigned to storage in the memory region called COMMON. The elements named are assigned storage relative to the common origin in the order of their appearance in the COMMON statement of each program unit. By making use of this positional relationship, more than one program unit in an executable program may reference the same data directly.

Each entity type (real or integer) is assigned two storage locations relative to the beginning of common, and entities of the same type in corresponding position are the same quantity. Entities referenced by position are the correct type, if the most recent value assignment to that position was of the same type.

The size of common in each program unit of an executable program may vary without disturbing the specified positional relationship. The beginning of common is established during the loading process with the program unit with the largest common region and all other program units are adjusted to begin at this location.

A program may have more than one COMMON statement, however, it may be preceded only by a FUNCTION, SUBROUTINE, DIMENSION or a prior COMMON statement.

Example:

```
DIMENSION      A(5)
COMMON         I, A, B
```

This example would result in the first element of COMMON being the integer variable I, the next five elements of COMMON being the real vector array A, and the next element in COMMON being the real variable B.

12.3.5 EQUIVALENCE STATEMENT.

Form: EQUIVALENCE (k), (k_2), $\dots$, (k_n), where each (k) is a list of two or more non-dummy variables and/or array element names, separated by commas. Subscript expressions of array element names must be non-zero, unsigned integer constants. A two dimensional array may be referred to by using a single subscript, giving the element number within the array, if desired.

The effect of the EQUIVALENCE statement is to cause the same area of memory to be shared by two or more entities. Each element of the K_i list is assigned the same (or a part of the same) storage area.

More than one EQUIVALENCE statement is permitted in a program, but it may only be preceded by a SUBROUTINE, FUNCTION, DIMENSION, COMMON or prior EQUIVALENCE statement.

Example:

```
DIMENSION      A(5), I1(3,3) B1(3)
COMMON         B, B1, B2
EQUIVALENCE    (X, A(2), Y), (B, C2,
                  F5), (I1(5), B2)
```

The effect of an EQUIVALENCE statement upon COMMON assignments, may be the lengthening of COMMON. This lengthening is permitted only if it increases COMMON in the same direction as additional COMMON elements would. Thus, in the example, the equivalence (B, I1(5)) would have been invalid. It is also invalid to equate two elements of the same array to each other.

12.4 EXPRESSIONS AND ASSIGNMENTS

12.4.1 ARITHMETIC EXPRESSIONS

An arithmetic expression is formed in FORTRAN syntax by a combination of operations and elements. The expression and its elements identify the expression to be type integer, real or double precision.

The arithmetic operators are shown in the following table:

Operator	Function
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

The arithmetic elements are described by the following statements:

Primary An arithmetic expression enclosed in parenthesis, a constant, a variable reference, an array element reference or function reference.

Factor A factor is a primary of the forms:

primary ** primary

Term A term is a factor of one of the forms:

term/factor
or
term*term

Signed Term A term immediately preceded by a + or - sign.

Simple Expression A term or two simple arithmetic expressions separated by a + or - sign.

Arithmetic Expression A simple expression or a signed term or either of the preceding, immediately followed by a simple expression.

A primary of any type may be exponentiated by an integer primary and the resulting factor is of the same type as that of the element being exponentiated. A real primary may be exponentiated by a real primary, and the resulting factor is of type real. A real or double-precision primary may be exponentiated by a real or double-precision primary. The resultant factor is of type real if both primaries are real, and otherwise of type double-precision. These are the only cases for which use of the exponentiation operator is defined. Table 12.4-1 gives the valid combinations for exponentiation.

By use of the arithmetic operators other than exponentiation, any admissible element may be combined with another admissible element of the same type.

A part of an expression is evaluated only if it is necessary to establish the value of the expression. The rules for formation of expressions imply the binding strength of the operators. The range of the subtraction operator is the term of the operator that immediately succeeds it. The evaluation may proceed according to any valid formation sequence. Use of an array element name requires the evaluation of its subscript. The type of the expression

Table 12.4-1. Exponent

	-	Real	Integer	Double Precision
Base	Real	Valid	Valid	Valid
	Integer	Invalid	Valid	Invalid
	Double Precision	Valid	Valid	Valid

in which a function reference or subscript appears does not affect, nor is it affected by, the evaluation of the actual arguments of subscript. An element whose value is not mathematically defined cannot be evaluated.

The following rules represent the derivation of all permissible expressions:

A variable, constant, or function standing alone is an expression.

A(1) JOBNO 217 17.26 Sqrt(A+B)

If E is an expression whose first character is not an operator, when +E and -E are expressions.

-A(1) +JOBNO -217 +17.26
-SQRT(A+B)

If E is an expression, the (E) is an expression meaning the quantity E taken as a unit.

(-A) -(+JOBNO) -(X+Y)
(A-SQRT(A+B))

If E is an expression whose first character is not an operator, and F is any expression, then: F+E, F-E, F*E, F/E and F**F are all expressions.

-(B(I, J)+SQRT(A+B(K, L)))
1.7E-2**(X+5.0)
-(B (1+E, 3*J+K) + A)

The mode of an expression may be either integer or real, and is determined by the modes of its elements, which must be the same with the following exceptions:

A real quantity can appear in an integer expression only as an argument of a function.

I+LFUNC (B)

An integer quantity can appear in a real expression only as an argument of a function, as a subscript or as an exponent.

AFUNC (I+2) A(I, J+1) B**N

The order of evaluation of expressions is established by the use of parentheses in the statement. If parentheses are not indicated, the following conventions of mathematics apply:

The hierarchy of operations, in order of precedence is: exponentiation, followed by multiplication and division, followed by addition and subtraction.

Within the same hierarchy of operations, evaluation proceeds from left to right.

Examples:

$X+Y*Z$	is interpreted as $X+(Y*Z)$
$W*X/Y*Z$	is interpreted as $((W*X)/Y)*Z$
$B**2-4.*A*C$	is interpreted as $(B**2)-((4.*A$ $(*C))$
$X-Y-Z$	is interpreted as $(X-Y)-Z$
$X/Y/Z$	is interpreted as $(X/Y)/Z$
$-X**3$	is interpreted as $-(X**3)$

12.4.2 ARITHMETIC ASSIGNMENTS AND REPLACEMENTS

The assignment statement is used to replace the value of a variable with the results of the evaluation of an expression.

Form: $v = e$, where v is any variable or array element name, and e is an arithmetic expression.

If the mode of the expression is different than the mode of the variable, the value of the expression will be converted to cause its mode to be compatible with the mode of the variable. Table 12.4-2 defines the rules for assignment of e to v .

12.5 CONTROL STATEMENTS

12.5.1 GENERAL

Each statement in a FORTRAN program is executed in the order of its appearance in the source program, unless this sequence is interrupted or modified by a control statement. This section of the manual describes the various control statements used in DATA 620/i FORTRAN.

12.5.2 GOTO STATEMENTS

GOTO statements transfer logical control from one section of a program to another.

Basic FORTRAN includes two forms of the GOTO statement; unconditional and computed.

Unconditional GOTO

An Unconditional GO is of the form: GOTO k , where k is a statement label reference.

Execution of this statement causes the statement identified by the label k to be executed next in sequence.

Example:

GOTO 72

71 $V7 = HQ(5) + Y**L$

:

:

72 $V7 = HQ(4) + X**J$

In this example, execution of the GOTO 72 statement causes statement number 71 and any succeeding statements to be by-passed. Execution is resumed with statement number 72.

Computed GOTO

The computed GOTO statement is of the form: GOTO ($k_1, k_2, \dots, k_n$), i , where the k 's are statement label references, and i is an integer variable reference.

Execution of this statement causes the statement identified by the statement label k_i to be executed next in sequence where i is the value of i at execution time. Valid execution of this statement is dependent upon the value of the integer variable such that 1 is less than or equal to i , and i is less than or equal to n .

Table 12.4-2. Rules for Assignment of e to v

v	e	Assignment Rule
Integer	Integer	Assign
Integer	Real	Fix & Assign
Integer	Double Precision	Fix & Assign
Real	Integer	Float & Assign
Real	Real	Assign
Real	Double Precision	DP Evaluate & Real Assign
Double Precision	Integer	DP Float & Assign
Double Precision	Real	DP Evaluate & Assign
Double Precision	Double Precision	Assign

NOTES

- (1) Assign means transmit the resulting value, without change, to the entity.
- (2) Real assign means transmit to the entity as much precision of the most significant part of the resulting value as real datum can contain.
- (3) DP evaluate means evaluate the expression according to the most precise rules, then DP float.
- (4) Fix means truncate any fractional part of the result and transform that value to the form of an integer datum.
- (5) Float means transform the value to the form of a real datum.
- (6) DP float means transform the value to the form of a double precision datum, retaining in the process as much of the precision of the value as a double precision datum can contain.

Example:

GOTO (98,405.3), n

Execution of the statement in the example will cause control to be transferred to the statement labeled 98,405 or 3 if the value of the variable integer n is 1, 2 or 3 respectively. If n contains an integer other than 1, 2 or 3, the results of the transfer cannot be predicted.

12.5.3 ARITHMETIC IF STATEMENT

It is often necessary to alter the logical flow of a program on the basis of the results of an arithmetic test. The IF statement is a conditional transfer that will execute this level of control, and is of the form:

IF (e) k_1 , k_2 , k_3

The arithmetic IF is a three-way transfer. Execution of this statement causes the expression (e) to be evaluated, following which, the statement identified by the label k_1 , k_2 , k_3 is executed next in sequence, as the value of (e) is less than zero, equal to zero or greater than zero, respectively.

Example:

IF (1) 10, 11, 12
10 V7 = HQ (5) + Y**L

GO TO 13
11 V7 = HQ (4) + X**J

GO TO 13
12 V7 = HQ (3) + X**L

13 Next Statement

In this example, execution of the IF (1) 10, 11, 12 statement causes one of the following actions: for a negative value of 1, statement number 10 is executed in sequence; for a zero value of 1, statement number 10 and any succeeding statements are by-passed and statement number 11 is executed; for a positive, non-zero value of 1, statements 10 through 11 and any statement following statement 11 are by-passed, and statement number 12 is executed.

12.5.4 CALL STATEMENT

The CALL statement causes a transfer of execution control to a subroutine type subprogram, and is of one of the forms: CALL s (a_1 , a_2 , ..., a_n) and CALL s, where s is the name of a subroutine and the a's are actual arguments that will replace the dummy arguments in the called subroutine. Arguments may be variable names, array element names, array names, or any other expression. They must, however, be indicated in order, number and type with the corresponding dummy arguments of the subroutine.

Execution of the call statement transfers control the designated subroutine. The arguments declared in the statement line are associated with the dummy arguments that are parameters of the executable statements of the subroutine. Control is then passed to the first executable statement of the called subroutine. Control will be returned to the first executable statement following the CALL statement upon execution of the RETURN statement in the subroutine. Examples of calling sequences to subroutines are shown below.

CALL TEST (A,1)
CALL EXIT

The first example will transfer execution control to the subroutine labelled TEST, and the inclusion of the parameters or arguments A and I in the subroutine. The second example will cause execution control to be transferred to the subroutine labelled EXIT. Any arguments required for execution of EXIT are self contained in the logic of the subroutine.

12.5.5 RETURN STATEMENT

The execution of a RETURN statement results in the exit from a subprogram, and is expressed in the form: RETURN.

A RETURN statement defines the logical end of a procedure subprogram, and therefore may appear only in a subprogram. Execution of the statement returns logical control to the current calling program unit. Each subprogram must contain at least one RETURN statement.

In the case of a subroutine subprogram, control is returned to the first statement immediately following the CALL statement that released control to the subroutine. In the case of a function subprogram, control is returned (with the value of the function available), to the statement that called the function subprogram.

12.5.6 CONTINUE STATEMENT

Form: CONTINUE.

The CONTINUE statement results in no action in an execution sequence, and therefore the statement has no effect upon the program. This statement serves as a program unit reference point.

Example:

```

IF (I) 10, 11, 12
10 V7 = HQ (5) + Y**L
      .
      .
GOTO 13
11 V7 = HQ (4) + X**J
      .
      .
GOTO 13
12 V7 = HQ (3) + X**L
      .
      .
13 CONTINUE

```

12.5.7 PAUSE STATEMENT

Form: PAUSE n or PAUSE, where n is an octal digit string of length from 1 to 4.

A PAUSE statement causes a temporary cessation of program execution, and displays PAUSE n (see section 8 for display format). The statement permits operator intervention for setup or control functions, such as changing data tapes. The computer executes a halt instruction delaying further execution until the operator selects the console RUN button. Execution will resume at the first executable statement following the PAUSE statement.

Example:

```

PAUSE 01

```

12.5.8 STOP STATEMENT

Form: STOP n or STOP, where n is an octal digit string of length from 1 to 4.

A STOP statement causes termination of program execution, and displays STOP n (see section 8 for display format). The program then terminates with a halt instruction.

Example:

STOP 0721

12.5.9 DO STATEMENT

The DO statement is used to control repetitive execution of a group of statements. The number of repetitions is dependent upon the value of a control variable. The statement assumes one of the forms: $DO\ n\ i = m_1, m_2, m_3$ and $DO\ n\ i = m_1, m_2$, where:

n is the statement label of an executable statement. This statement, called the terminal statement of the associated DO must physically follow and be in the same program unit as the DO statement. The terminal statement may not be a GOTO of any form, arithmetic IF, RETURN, STOP, PAUSE or another DO statement.

Symbol i is an integer variable name, identified as the control variable.

Symbol m_1 , identified as the initial parameter, m_2 , as the terminal parameter, and m_3 , as the incrementation parameter are each either an integer constant or integer variable reference. If the second form of the DO statement is used, a value of 1 is implied for the incrementation parameter. When the DO statement is executed, the values of m_1 , m_2 and m_3 must be greater than zero.

Associated with each DO statement is a range that is defined to be those executable statements from and including the first executable statement following the DO, to and including the terminal statement defined by the DO. A special situation occurs when the range of a DO contains another DO statement. In this case, the range of the contained DO must be a subset of the range of the containing DO.

The control variable is assigned the value represented by the initial parameter. This value must be less than or equal to the value represented by the terminal parameter.

The range of the DO is executed.

If control reaches the terminal statement, and after execution of the terminal statement, the control variable of the most recently executed DO statement associated with the terminal statement is incremented by the value represented by the associated incrementation parameter.

If the value of the control variable after incrementation is less than or equal to the value represented by the associated terminal parameter, the action is repeated with the understanding that the range in question is that of the DO, the control variable of which was most recently executed.

If the value of the control variable is greater than the value represented by its associated terminal parameter, the DO is said to be satisfied, and the control variable becomes undefined.

If there were one or more other DO statements referring to the terminal statements in question, the control variable of the next most recently executed DO statement is incremented by the value represented by the associated incrementation parameter until all DO statements referring to the particular termination statement are satisfied, at which time the first executable statement following the terminal statement is executed.

Upon exiting from the range of a DO by execution of a GOTO statement or an arithmetic IF statement, that is other than by satisfying the DO, the control variable of the DO is defined and is equal to the most recent value attained.

A GOTO statement or an arithmetic IF statement may not cause control to pass into the range of a DO from outside its range. When a procedure reference occurs in the range of a DO, the actions of that procedure are considered to be temporarily within that range, i.e., during the execution of that reference.

The control variable, initial parameter, terminal parameter, and incrementation parameters of a DO may not be redefined during the execution of the range of that DO.

If a statement is the terminal statement of more than one DO statement, the label of that terminal statement may not be used in any GO TO or arithmetic IF statement that occurs anywhere but in the range of the most deeply contained DO with that terminal statement.

Example:

```
DO 607 K1 = 2, ID, 3
```

The foregoing statement would cause K1, the control variable, to be set to the value of the initial parameter, 2. Execution would proceed at the statement immediately following, down to and including the statement identified by the label 607. After each execution of the loop, K1 is incremented by the incrementation parameter, 3, and evaluated in relation to the current value of the terminal parameter, ID. If the current value of ID is greater than K1, execution control is transferred to the statement following that identified by the label 607, otherwise the DO cycle is repeated.

12.6 INPUT/OUTPUT STATEMENTS

12.6.1 GENERAL

Input statements provide a program with the means of receiving information from external

sources. Output statements allow the transmission of program data to extend sources. These external sources may be devices such as magnetic tape and paper tape handlers, typewriters, and punch card processors.

There are two types of input/output statements.

- a. READ and WRITE statements
- b. Auxiliary statements

The first type cause the transfer of records of sequential files to and from the program. This data may be formatted information consisting of strings of characters, or unformatted information consisting of binary word values in the form in which they normally appear in storage. The second statement type consists of the BACKSPACE and REWIND statements which provide for positioning of magnetic tapes, and the ENDFILE statement which provides for closing of a file.

Input/output statements reference input/output units and, formatted information, format specifications. An input/output unit is identified by a logical unit number, *u*, which may be either an integer constant or a variable name that references an integer constant. Logical unit number assignments for the DATA 620/i FORTRAN may be found in appendix H-2. The format specification is defined by a FORMAT statement having the statement label *f*. This statement must appear in the same program as the input/output statement.

12.6.2 INPUT/OUTPUT LISTS

The input list specifies the names of variables and array elements to which input values are assigned. The output list specifies the names of variables and array elements whose values are transmitted. Input and output lists are of the same form.

12.6.3 SIMPLE LISTS

Simple lists have the form: $m_1, m_2, m_3 \dots, m_n$ where the m_i are the names of real or integer variables or array elements. The comma characters separate each individual name in the list. The period characters signify possible additional list items. List elements may be enclosed in parentheses.

Example:

Input Lists	Output Lists
A C (26, L) R, K, D, (I, J)	B I (10, 10) S, (R, K), F (1, 25)

An array variable which is not subscripted in a list is considered equivalent to the listing of each successive element of the array. If B is an array, the list B is equivalent to B (1, 1), B (2, 1), B (3, 1), ..., B (1, 2), B (2, 2), ..., B (j, k), where j and k are the subscript limits of B.

12.6.4 DO-IMPLIED LISTS

A DO-implied list is a simple list followed by a comma character and an expression of the form: $i = m_1, m_2, m_3$ or $i = m_1, m_2$.

The elements i, m_1, m_2 and m_3 have the same meaning as defined for the DO statement. The DO implication applies to all simple list items enclosed in parentheses with the implication. For input lists, i, m_1, m_2 and m_3 may appear within this range only as subscripts.

Examples:

(X (I), I = 1, 4)
(Q (J), R (J), J = 1, 2)
(G (K), K = 1, 7, 3)
((A (I, J), I = 3, 5), J = 1, 2)

(X (K), K = 1, 2), I, (R (J), J = 3, 5)
X (1), X (2), X (3), X (4)
Q (1), R (1), Q (2), R (2)
G (1), G (4), G (7)
A (3, 1), A (4, 1), A (5, 1),
A (3, 2), A (4, 2), A (5, 2)
X (1), X (2), I, R (3), R (4), R (5)

12.6.5 READ STATEMENTS

These statements are used to obtain data values from an external source. The data values are input in either formatted or unformatted mode. The form of a formatted READ statement is: READ (u, f) k.

The verb READ and the parentheses must appear in this form.

Execution of this statement causes information to be transmitted from the external source whose logical unit number is defined by u. This data is scanned and converted as specified by the format specification, f, and the resulting values are assigned to the variable names defined in the list, k.

The form of an unformatted READ statement is: READ (u) k.

The verb READ and the parentheses must appear in this form. This statement causes data to be input in binary form from the unit defined by u. The values are assigned to the variable names defined in the list, k.

Examples:

READ (1, 44) A, B, C
READ (2) R, S
READ (N, 12) A, (R (I), I = 1, 10)
READ (L) S, (T (J), J = 1, N)

All information appearing on external sources is divided into records. Each time a READ statement is executed a new record is processed. The number of records input by a single READ statement is determined by the list and format specification. If only part of a record is input the remainder of the record is lost as the next READ processes the next record. Records are read sequentially until the list is exhausted. Only enough values are read to fill the list.

The list, *k*, in an unformatted read statement may be left blank to skip a record.

The record size for formatted data is 80 characters except when the device is the Teletype keyboard or paper tape in which case the record size is variable with a maximum of 80 characters processed per record. Unformatted records are 64 binary words in length.

12.6.6 WRITE STATEMENTS

WRITE statements are used for the purpose of transferring program data to external devices. This data may be formatted or unformatted. The form of a formatted WRITE statement is: WRITE (*u*, *f*) *k*.

The verb WRITE and the parentheses must appear in this form.

Execution of this statement causes records to be written on the device referenced by *u*. The contents of the records are the values taken sequentially from the list *k* converted according to the format specification *f*.

The form of an unformatted WRITE statement is: WRITE (*u*) *k*.

The verb WRITE and the parentheses must appear in this form.

Execution of this statement causes binary information from the list *k* to be written in records on the unit defined by *u*.

Example:

```
WRITE (1, 4) A,B,C
WRITE (7) R,S,T
WRITE (K, 12) X,(Y(J), J=1,M),I
WRITE (N) W, Z, (F(K), K=1, 5)
```

Several records may be written with a single WRITE statement. The number of records is determined by the list and the format specifications. Successive records are written until the data is exhausted. If the data does not fill a record, the record is filled with blanks.

12.6.7 REWIND STATEMENT

This statement is of the form: REWIND *u*.

Execution of this statement causes the magnetic tape unit defined by *u* to be rewound. If *u* is not a magnetic tape, no action is taken.

12.6.8 BACKSPACE STATEMENT

This statement has the form: BACKSPACE *u*.

The BACKSPACE statement causes the magnetic tape unit defined by *u* to be backspaced one record. If *u* is not a magnetic tape, no action is taken.

12.6.9 ENDFILE STATEMENT

This statement has the form: ENDFILE *u*.

When this statement is executed, a file mark is written on the magnetic tape defined by *u*. No action is taken if *u* is not a magnetic tape.

12.6.10 FORMAT STATEMENTS

FORMAT statements are used with input-output operations to specify conversion and editing of information between program storage and external representation. FORMAT statements are non-executable and must have a statement label to be referenced by input/output statements. Conversion performed according to a FORMAT statement during output is in general the reverse of conversion performed during an input operation.

A FORMAT statement is expressed as: n FORMAT ($f_1, f_2, f_3, \dots, f_n$), where n is the statement label and the f_i are field specifications. The noun FORMAT and the parentheses must appear in this form. The comma characters are required only when ambiguities would arise from not separating field specifications. The period characters signify possible additional field specifications and would not actually be present.

12.6.11 FIELD SPECIFICATIONS

Field specifications describe the type of conversion and editing to be performed on each variable appearing in the input-output list. Field specifications may be any of the following forms:

rAw
rFw.d
rEw.d
rDw.d
rlw
nHs
nX

where:

- The characters A, D, E, F and I indicate the manner of conversion for variables in the list.

- The characters H and X represent character to be input-output directly from the format.
- The character / represents the end of a record.
- w and u are non-zero integer constants defining the width of the field (including digits, decimal points, algebraic signs) in the external character string.
- d is an integer specifying the number of fractional digits appearing in the external string.
- r is an optional, non-zero integer indicating that the specification is to be repeated r times.
- s is a string of acceptable FORTRAN characters.

12.6.12 F CONVERSION

Form: rFw,d

Only real data may be processed by this form of conversion.

Output

The field is right justified with as many leading blanks as necessary to fill w. Negative values are preceded by a minus sign. Internal values are converted to fixed point decimal numbers and rounded to d decimal places.

For a field specification of F10.4:

368.4	is converted to	368.4000
12.0	is converted to	12.0000
-17.90767	is converted to	-17.9077
37.5E-2	is converted to	0.3750

If a value requires more positions than allowed by w the most significant digits, including sign if negative, are output. The error indication is designated by an asterisk in the least significant character position.

For a field specification of F6.4:

4739.76	is converted to	4740.0*
-12.463	is converted to	-12.5*

Input

Input strings are decimal numbers of length w with d characters in the fractional portion. Blanks are treated as zeros. If a decimal point is present in a value the fractional portion of the value is explicitly defined by that decimal point character. A common (,) terminator may be used to override the w specification. Terminated fields are treated as normal fields with leading zeros. A comma alone defines a zero value for the field.

For a field specification F8.3:

35	is converted to	0.035
964372	is converted to	964.372
0.53821	is converted to	0.53821
-16.402	is converted to	-16.402
-12	is converted to	-0.012
47.E-4	is converted to	0.0047
36,	is converted to	0.036
-0.75,	is converted to	-0.75
,	is converted to	0.0

12.6.13 E CONVERSION

Form: rEw.d.

Only real data may be processed by this form of conversion.

Output

Internal values are converted to decimal values of the forms: .ddd...dE ee and .ddd...dE-ee, where ddd...d represent d digits, while ee is a decimal exponent. The leading decimal point and E characters are present exactly as shown. Internal values are rounded to d digits and negative values are preceded by a minus sign. The external field is right justified and preceded by blanks to fill the width, w. This field width includes the exponent digits, the sign of the exponent (minus or space), the letter E, the magnitude digits, the decimal point, and the sign of the value (minus or space). This means that the field width should correspond to the relation: $w \geq d + 6$.

If w is less than (d + 6) the format is in error.

For the field specification E12.5:

76.573	is converted to	0.76573E 02
58796.341	is converted to	0.58795E 05
-369.7583	is converted to	-0.36976E 03
0.006873	is converted to	0.68730E-02
0.2	is converted to	0.20000E 00
-0.0000054	is converted to	-0.54000E-05

Each external value is of field width w with d characters in the fractional part of the value. The value is right justified with all blanks counting as zeros. A minus sign may be placed preceding the value of the exponent. A decimal point placed in the fractional part takes precedence over the d specification. The character E should be present to separate the value and the exponent. If not, the exponent is taken as the two least significant digits. A comma (,) terminator may be used to override the w specification. Terminated fields are treated as normal fields with leading zeros. A comma alone defines a zero value for the field.

For a field specification E10.3:

123E3,	is converted to	123.0
12874E2	is converted to	1287.4
-563E-02	is converted to	-0.00563
-6.7563E05	is converted to	-675630.0
398E00	is converted to	0.398
5387601	is converted to	538.76
5455-01	is converted to	0.5455

12.6.14 D CONVERSION

The D conversion is used for the input/output of double-precision numbers. It is used exactly as the E conversion except the letter E is replaced by D.

12.6.15 I CONVERSION

Form: rlw

Only integer data may be processed by this form of conversion.

Output

Internal values are converted to integer constants. Negative values are preceded by a minus sign. Each field is right justified and filled with leading blanks.

For the field specification 16:

281	is converted to	281
-43567	is converted to	-43567

If the data requires more character positions than allowable by the width w, only the least significant w positions are output.

For the field specification 12:

281	is converted to	81
-6374	is converted to	74

INPUT

External input values are right justified with the width w. Blanks are counted as zeros. Input values must be integer values. A preceding minus sign may be placed on a value. A comma (,) terminator may be used to override the 10 specification. Terminated fields are treated as normal fields with leading zeros. A comma alone defines a zero value for the field.

For the field specification 14:

120	is converted to	120
-144	is converted to	-144
1 2	is converted to	1020
-3,	is converted to	-3

12.6.16 A CONVERSION

An A Format conversion is used in conjunction with a READ or WRITE statement for the input/output of alphanumeric information to or from a REAL list element. The general form is nAw where n and w are both unsigned integer constants. If n is one, it may be omitted.

On input, nAw will be interpreted to mean that the next n successive fields of w characters each are to be stored in the associated REAL list elements. If w is greater than 4, only the 4 right-most characters will be significant. If w is less than 4, the characters will be left-adjusted, and the word (s) filled out with blanks.

On output, nAw will be interpreted to mean that the next n successive fields of w characters each are to be the result of alphanumeric transmission from the specified REAL list elements. If w exceeds 4, only 4 characters of output will be transmitted, preceded by w-4 blanks. If w is less than 4, the w left-most characters of the specified storage element will be transmitted.

12.6.17 H CONVERSION

In DATA 620/i FORTRAN, Hollerith information consists of the legal FORTRAN character set plus the additional characters \$, !, ", #, %, &, ', :, ;. Information input from the typewriter or paper tape is converted to an internal code used by FORTRAN. When this information is output, the internal codes are converted to the appropriate typewriter or paper tape codes.

Form: wHs.

Output

The number of characters, w, in the string, s, should contain exactly the number of characters specified so that characters from other fields are not taken as part of the string.

Blanks are counted as characters in the string.

Examples:

Specification	External Output
1 HR 8H STRING 12HX (1,3) = 12.0	R STRING X (1, 3) = 12.0

Input

The w characters in the string s are replaced by the next w characters from the input record. The resultant is a new string in the field specification.

For Example:

Specification	Input String	Resultant Specification
5H12345 7H TRUE 8H	ABCDE FALSE MATRIX	5HABCDE 7HFALSE 8HMATRIX

This feature can be used to change titles, dates, headings, etc., which are output with the program data.

12.6.18 X SPECIFICATION

Form: wX.

This specification causes no conversion to occur. On output, w blanks are inserted in the external record. On input, w spaces are skipped from the input record.

Example of output:

Specification	Output
1HA, 4X, 2HBC 4X, 3HABC 1X, 3HABC, 3X	A BC ABC ABC

Example of input:

Specification	Input String	Resultant Input
F4.1, 3X, F3.0	12.5RRR120	12.5,120.

The RRR characters are ignored by the 3X specification.

12.6.19 / SPECIFICATION

Form: /.

Each slash (/) specified in the format causes the termination of a record and processing of the next record. Successive slashes (/... /) cause successive records to be ignored on input, and successive blank records to be written on output. A slash separating two field specifications removes the need for a comma separator.

For example:

F5.4,/4F10.3 is equivalent to
F5.4/4F10.3

Output Example:

For a specification (1HA/1HB/1HC/1HD)
the resultant output records are:

A
B
C
D

Input Example:

Using the four records output from the previous example, an input specification (1H1/1H2//1H3) produces the resultant specification (1HA/1HB//1HD).

12.6.20 REPEAT SPECIFICATIONS

The A, D, F, E and I field specifications may be repeated by using the repeat count *r* in the forms *rAw*, *rDw*, *rFw.d*, *rEw.d* and *rlw*.

Examples:

4F10.5, F3.6 is equivalent to F10.5,
F10.5, F10.5, F10.5, F3.6

2F4.1, 2E7.1 is equivalent to F4.1,
F4.1, E7.1, E7.1

2F5.2, 316, 2E8.2 is equivalent to F5.2,
F5.2, 16, 16, E8.2, E8.2

Repetition of a group of field specifications is accomplished by enclosing the group in parentheses preceded by an integer repeat count. If no repeat count is specified the count is taken as one.

Examples:

2(F10.5, 16) is equivalent to F10.5, 16,
F10.5, 16

2(E9.3, F7.1/14) is equivalent to E9.3,
F7.1/14, E9.3, F7.1/14

3(4F5.0, 2E8.2) is equivalent to 4F5.0,
2E8.2, 4F5.0, 2E8.2, 4F5.0, 2E8.2

Example:

50 FORMAT (4X, 2(15,6F8.2)/3(312.7,
F6.4), 214)

12.6.21 FORMAT CONTROL AND LIST INTERACTION

Execution of a formatted READ or WRITE statement initiates format control. The conversion performed on data depends on information jointly provided by the next element of the input-output list and the next field specification of the FORMAT statement. If there is a list, at least one field specification of type D, E, F or I should be present in the FORMAT statement.

Execution of a formatted READ statement causes one record to be input. To each D, E, F or I specification there corresponds one element in the list. To each H or X specification there is no corresponding element in the list and the format control communicates information directly with the record. Whenever a slash is encountered, or the entire input record is processed, the record is terminated. If more input is necessary the next record is input. Any unprocessed characters of a record are skipped when a slash is encountered.

If specification is A, list is input or output directly from storage.

A READ statement is terminated upon expiration of the list if:

- a. the next specification is a D, E, F or I;
- b. the format control has reached the last outer right parenthesis of the FORMAT statement.

If the list expires and the next specification is an H or X, data is processed (with the possibility of additional records being input) until one of the above two conditions is met.

If the format control reaches the rightmost parenthesis of the FORMAT statement and more list remains to be processed the following steps are taken:

- a. a new record is input and any remaining data in the previous record is ignored;
- b. format control reverts to the point immediately following the last left parenthesis encountered.

If group repeat specifications exist in the format, this point is at the rightmost group of the format. The repeat count is not taken into consideration. If no groups are present, the format is started from the beginning.

When a formatted WRITE statement is executed, records are written each time 80 characters or (72 characters in the case of teletype keyboard records) have been processed, a slash is encountered, or the format control terminates. The format control terminates by one of the two methods described for READ termination. Incomplete records are filled with blanks to maintain standard record lengths.

12.7 PROGRAMS AND SUBPROGRAMS

12.7.1 GENERAL

An executable FORTRAN program consists of a main program and any required subprograms. Subprograms may be defined by the programmer or may be contained in the system library. Each program or subprogram must contain at least one executable statement.

12.7.2 MAIN PROGRAMS

A main program is a program unit consisting of a set of FORTRAN statements, comment lines, and an END line. The program may be preceded by specification statements. If so, these statements must be in the following order: DIMENSION, COMMON and EQUIVALENCE.

A main program cannot contain a subprogram definition statement, namely:

- a FUNCTION statement
- a SUBROUTINE statement

A main program may contain calls to other subprograms or may contain statement function subprograms.

12.7.3 SUBPROGRAMS

Subprograms are program units which may be called by other programs or subprograms. Subprograms are categorized as one of the following:

- Statement functions
- Intrinsic functions
- FUNCTION subprograms
- SUBROUTINE subprograms

The first three are categorized as functions and the last as subroutines.

Functions are programmed procedures which are often used to provide solutions to mathematical functions. Function references may be used in the same manner as references to variables in an expression. For example: $X = AB * \sin(Y) - C * \cos(Y * Z)$, where SIN is the name of the sine function, COS is the name of the cosine function, and (Y) and (Y*Z) are their respective argument lists. The value returned for a function reference is of the same mode as the function name, corresponding to the rules for real and integer symbolic names.

12.7.4 STATEMENT FUNCTIONS

A statement function is defined internally to the program unit in which it is referenced. All statement functions must precede the first executable statement and must follow any specification statements of the program unit.

A statement function is defined in a single expression of the form: $f(a_1, a_2, a_3, \dots, a_n) = e$, where f is the function name, the a_i are the arguments, and e is an expression. The resultant value of the function is either a real or integer value corresponding to the function name. The a_i are distinct variable names and are called dummy arguments. These serve to indicate the type, number, and order of the function arguments. The expression e is an arithmetic expression and may contain references to previously defined statement functions.

A statement function is referenced by a function call, $f(a_1, a_2, a_3, \dots, a_n)$, appearing in an arithmetic expression. A statement function may only be referenced within the program unit in which it is defined. The arguments used in the reference must agree in type, number, and order with the corresponding dummy arguments.

Example:

The statement function:

$$SF(X) = A * X ** 2 + B * X + C$$

may be referenced in the program by:

$$W = SF(Y)$$

12.7.5 INTRINSIC FUNCTIONS

Intrinsic functions are commonly used subprograms and are contained in the FORTRAN library. The symbolic names and meanings of the intrinsic functions are shown in Appendix H-5.

An intrinsic function is referenced by a function call in an arithmetic expression. The arguments in the argument list must agree in type, number and order with those shown in Appendix H-5.

Example:

```
IF (SIGN(W,X)) 1,2,2,
1 W=ABS(X) - ABS(Y)
2 S=W*FLOAT(I*J)
K=IFIX(X) + J
```

12.7.6 FUNCTION SUBPROGRAMS

A function subprogram is defined externally to the program unit by which it is referenced. A function subprogram is defined by having as its first statement, other than comment lines, a statement of the form:

FUNCTION $f(a_1, a_2, a_3, \dots, a_n)$

where f is the symbolic name of the function and the a_i are dummy arguments. Each a_i is either a variable name or an array name. The

a_i define the type, number and order of the FUNCTION arguments.

A function subprogram is executed at the first executable statement following the FUNCTION statement. Specification statements (DIMENSION, COMMON and EQUIVALENCE) may immediately follow the FUNCTION statement. If present, these must precede any other statement, excluding comments. The symbolic names of the dummy arguments, a_i , may not appear in an EQUIVALENCE or COMMON statement.

A function subprogram must contain at least one RETURN statement and the last statement executed in a FUNCTION must be a RETURN statement. The function subprogram is ended by an END line.

The symbolic name, f , of the FUNCTION must appear as a variable name within the subprogram. The value returned for a FUNCTION is the last value assigned to this name prior to execution of a RETURN statement. The mode of the FUNCTION value, either integer or real, is determined from the function name.

The symbolic name of the function must not appear in any nonexecutable statement within the subprogram. A subprogram may not define or redefine any of its arguments nor any variable in COMMON.

Example FUNCTION:

```
FUNCTION XP(A,B,1)
  DIMENSION B(10)
  XP=(A*B(J)**1)+XP
  DO 1 J=1, 10
1  XP=(A*B(J))**1+XP
  RETURN
END
```

A FUNCTION is executed with a function reference by a main program or another subprogram. The actual arguments in the call must correspond in type, number, and order with the FUNCTION dummy arguments. If a dummy argument of a FUNCTION is an array name the corresponding actual argument must be an array name.

Example:

A call for the example FUNCTION shown above would be: $W+XP(R,S,K)$ where S is an array.

12.7.7 BASIC EXTERNAL FUNCTIONS

Basic external FUNCTIONS are standard subprograms contained in the FORTRAN library. These are referenced in the same manner as normal FUNCTIONS. The symbolic names and meanings of the basic external FUNCTIONS are shown in Appendix H-6.

12.7.8 SUBROUTINE SUBPROGRAMS

A subroutine subprogram is defined externally to the program unit that references it. Subroutines, unlike functions, do not have values associated with them and cannot be referenced in an expression. Subroutines are accessed by CALL statements.

A subroutine subprogram is defined by having as its first statement, other than comment lines, a statement of the form: SUBROUTINE $S(a_1, a_2, a_3, \dots, a_n)$ or SUBROUTINE S , where S is the symbolic name of the subroutine and the a_i are the dummy arguments of the subroutine. Each a_i is either a variable name or an array name. If no arguments are passed to the subroutine the second form above is used.

The symbolic name of the subroutine must not appear in any statement in the subprogram. The symbolic names of the dummy arguments may not appear in COMMON or EQUIVALENCE statements.

A subroutine is executed at the first executable statement. Specification statements may be contained immediately following the SUBROUTINE statement and preceding any executable statement. A subroutine must have at least one RETURN statement. The last statement executed by a subroutine must be a RETURN statement.

DATA 620/i FORTRAN includes a subroutine named EXIT. When this subroutine is referenced by a CALL statement of the form:

CALL EXIT

The statement END OF JOB will be displayed (see section 8 for display format), and the program terminates with a halt instruction.

Example SUBROUTINE:

```
SUBROUTINE R(A,I,Z)
  DIMENSION A (10)
  Z=0
  DO 1 J=1, 10
1  Z=Z+A(J)**1
  RETURN
  END
```

A subroutine is referenced with a CALL statement. The argument list in the reference must agree in type, number and order with the dummy arguments of the subroutine. If a dummy argument is an array name, the corresponding actual argument must be an array name.

Example:

A call for the example SUBROUTINE above would be: CALL R (T,K,D) where T is an array.

12.7.9 DUMMY ARGUMENTS

Dummy arguments provide a means of passing information between a subprogram and the program or subprogram which called it. Both function and subroutine subprograms may have dummy arguments. A subroutine need not have any, while a function must have at least one. Dummies provide definitions of the data type, number and sequence of subprogram parameters.

A dummy may be classified within a subprogram as a variable or an array. The actual arguments defined by a calling program or subprogram to which a dummy may correspond are: variables, array elements, arrays, expressions.

Within a subprogram a dummy may be used in much the same way as any other variable or array. A dummy may not appear in a COMMON or EQUIVALENCE statement.

The actual arguments used in a calling statement must agree in data type with the corresponding dummy arguments, that is, reals to reals, integers to integers, and arrays to arrays. If an actual argument is an expression, the result of the expression should correspond in data type to the dummy.

A dummy array is defined to be an argument which appears in DIMENSION statement in the subprogram. A dummy array does not occupy any storage but tells the subprogram that the argument supplied in the calling statement defines the first element of an

actual array. The calling argument need not have the same dimensions as the dummy array. Useful operations can sometimes be performed by defining different dimensions for the dummy and calling arguments.

Example:

DIMENSION	A(10,10)
CALL	FM(A(6,1))
SUBROUTINE	FM(B)
DIMENSION	B(50)

For this case the 1, dimensional dummy array B corresponds to the last half of the 2, dimensional array A. If the calling statement were: CALL FM(A).

The dummy array B would correspond to the first half of the array A.

12.8 FORTRAN OPERATING INSTRUCTIONS

12.8.1 GENERAL

The DATA 620/i FORTRAN system operates in a minimum configuration of 8192 words of memory and an ASR-33/35 teletype. FORTRAN programs and subprograms are compiled by the basic FORTRAN compiler. FORTRAN compatible machine language subprograms are assembled by the DAS assembler version I, Mod F. The FORTRAN loader loads main programs and all required subprograms into memory for execution. The FORTRAN run-time library provides input/output, control, and mathematical functions required at execution time.

12.8.2 COMPILER OPERATING INSTRUCTIONS

The DATA 620/i FORTRAN compiler translates FORTRAN source programs to relocatable machine language programs in a single pass.

FORTAN statements may be input from the teletype keyboard or paper tape reader, the card reader, the high speed paper tape reader or magnetic tape. Object code is output via the teletype or high speed paper tape punch or magnetic tape. Error diagnostics, source listings and object listings are provided on the teletype or line printer. Input/output and listing options are selected at the teletype keyboard for each program to be compiled.

12.8.3 PRELIMINARY OPERATIONS

The DATA 620/i FORTRAN compiler is supplied as an absolute binary object tape. The compiler is loaded into memory by the standard binary loader and occupies the first 13500 (8) words of memory. (See section 10.7 for procedure to load absolute object programs.) Entry to the compiler is at location 0. Upon entry, the compiler will execute a HALT 0777 with the A register set to the upper limit of compiler used memory (015777 standard for 8K). This limit may be modified by resetting the A register. (See appendix H-3 for compile time memory map.) To compile programs press RUN.

12.8.4 NORMAL OPERATIONS

For each program to be compiled a ?= will be typed on the teletype printer requesting input/output selection. The operator should respond by typing one of the following characters to indicate the input device: C (card reader), K (teletype keyboard), P (paper tape), 0 through 3 (magnetic tape, units 0 through 3); followed by one of the following characters to indicate the output device: C (card punch), P (paper tape), 0 through 3 (magnetic tape, units 0 through 3); followed by an (optional) listing selection character: S (source listing), Ø (object listing), B (both source and object

listings); followed by the character >, followed by the (optional) 1 to 6 character program name, followed by @ for a carriage return and line feed.

Example:

? = CPS > MATRIX @

C for input cards, P for output paper tape, S for list source with program name MATRIX. Following input/output selection, source statements are read and object records are output through the selected devices. Error diagnostics and selected list options are printed on the teletype or line printer (if available). Upon detecting an END statement (followed by a non-blank statement), the compiler will produce a program map listing all variables, constants (in octal) and required subprograms. Having listed the program map the compiler will type a ? = to permit compiling another program.

12.8.5 INPUT RECORDS

Input to the compiler is a series of FORTRAN statements each of which appear in one or more input records. Records may be fixed or variable in length depending on the device, however, only the first 72 characters of each record are used by the compiler. Any illegal characters are treated as blanks. Blank records are ignored. END statements must be followed by at least one non-blank record (another END statement is suggested).

Keyboard and paper tape records are variable length and are terminated by a carriage return and line feed in that order. The character > may be used to TAB to column 7, and the character ← may be used to clear the input buffer and reset to column 1. For keyboard input the teletype bell is used to notify the operator that source input is required.

Card records are a fixed length of 80 characters. The special characters > and ← are treated as blanks.

Magnetic tape records are a fixed length of 84 characters, and should be card images with blank padding characters. The special characters > and ← are treated as blanks. Carriage return and line feed characters are permitted but ignored.

12.8.6 OUTPUT RECORDS

Object records are a fixed length of 64 words and are output from time to time as they are created. Paper tape object programs are punched with leader and trailer records.

Magnetic tape object programs are terminated by an end of file. All main programs are terminated by an end-of-tape record. Refer to Appendix H-4 for object record format.

All error diagnostics are of the form: ERR xx a...a, where xx is a number from 1 to 15 (notification error) or T followed by a number from 1 to 9 (terminating error), and a...a represents the last (up to 16) characters encountered in the statement being processed. The right most character indicates the point where the error was discovered (the character @ indicates end of statement). If a terminating error is discovered object output is terminated, but source code is continued to detect any further errors.

12.8.7 NOTIFICATION ERRORS

1. Construction.
2. Usage.
3. Mode.
4. Illegal DO termination.
5. Improper statement number.
6. Common base lowered.

7. Illegal equivalence group.
8. Reference to non-executable statement.
9. No path to this statement.
10. Multiply defined statement number.
11. Invalid format construction.
12. Spelling error.
13. Format with no statement number.
14. Function not used as variable.
15. Truncated value.

12.8.8 TERMINATING ERRORS

- T1. Construction.
- T2. Usage.
- T3. Data pool full.
- T4. Illegal statement.
- T5. Improper use of name.
- T6. Improper statement number.
- T7. Mode.
- T8. Constant too large.
- T9. Improper DO nesting.

12.8.9 OPTIONAL LISTINGS

Source and object records may be listed if desired. Source records are listed as they are input. Object records are listed from time to time as they are created. Each object record consists of a varying number of 2 and 4 word data/instruction entries. The object record listing consists of one line for each entry. Two word entries are of the form `abbc vvvvvv`, and four word entries are of the form `abbc nnnnnn vvvvvv`, where `a` is the control code, `bb` is the sub code, `c` is the pointer number, `nnnnnn` is a 1 to 6 letter subprogram name and `vvvvvv` is a 6 digit octal value or instruction. See Appendix H-4 for object record format and codes.

12.8.10 PROGRAM MAP

Upon processing the END statement the compiler will list the program map. The first

three lines of the map define the size of the program, data and common areas and are of the form `a, *SIZE mmmmmm`, where `a` is the area (0 = program, 1 = data, 2 = common) and `mmmmmm` is the octal size. For programs with no terminating errors the following information is also listed.

a. <code>a, 11111 nnnnn</code>	Variable
b. <code>a, 11111 ccccc ccccc</code>	Constant
c. <code>S, 11111 nnnnn</code>	External subprogram
d. <code>#, 11111 ssss</code>	Statement number
e. <code>X, 11111 ssss</code>	Undefined statement number

Where `a` is the area, `11111` is the relative location of the item or the last reference to the subprogram or statement number, `cccccc ccccc` is a two word octal constant and `ssss` is a statement number.

12.8.11 FORTRAN LOADER OPERATING INSTRUCTIONS

The FORTRAN loader is designed to operate in a DATA/i 620 computer with at least 8192 words of memory. Its function is to load relocatable object programs produced by the DATA 620/i FORTRAN compiler and FORTRAN compatible subprograms produced by the DATA 620/i assembler. Object program input is from either paper or magnetic tape and is selected from the teletype keyboard. Load maps and error diagnostics appear on the teletype printer. See Appendix H-3 for load time memory map.

12.8.12 PRELIMINARY OPERATIONS

The FORTRAN loader is supplied as an absolute binary object tape and is loaded into memory by the binary loader (see section 10.7 for loading procedure). The FORTRAN loader occupies locations 000 through 077 and 0X4000 through 0X5740.

(Locations 0200 through 0377 are reserved for loader generated pointers.) The first program to be loaded must be a FORTRAN compiled main program. Prepare the input unit, clear the registers and RUN at location STRT (0X4140). The message 1N will appear on the teletype, requesting input selection. To select paper tape input type P. To select magnetic tape, type the unit number (0, 1, 2 and 3). If the selected unit is attached and ready the loader will load the main program (at location 0500). The teletype will then type RQ followed by a list of the subroutine required, followed by another input selection request.

12.8.13 LOADING SUBPROGRAMS

To effect the most efficient use of the loader, it is recommended that subprograms be loaded in the following order:

- a. Run-time input/output.
- b. Double-precision math functions.
- c. Single-precision math functions.
- d. Double-precision math library.
- e. Single-precision math library.
- f. Run-time utility.

Prepare and select the input unit as for main programs. The loader will load all required subprograms until an end of tape record is detected, at which time the list of required subprograms is generated and input selection is again requested. (NOTE: The end-of-tape

record is not produced for subprograms by either the compiler or the assembler, but is supplied as a separate tape labeled FORTRAN END-OF-TAPE, and should be spliced on the end of the users subprogram library tapes. Standard FORTRAN library tapes are delivered with end-of-tape records.)

If two or more subprograms have the same name, only the first such subprogram input will be loaded. When all required subprograms have been loaded, the message GO will be typed followed by the load map, which lists each subprogram loaded and its entry point. To execute the loaded program press RUN. The load map may be forced by running of symbolic location MAP in the loader. Execution of the main program may be forced by running at location 0500.

12.8.14 ERROR DIAGNOSTICS

An error in the loading process will cause type out of an error message and the load map. A minus sign will precede the address of each subprogram which has not been loaded, in this case, the address represents the last location at which the subprogram is requested. All errors except checksum errors are nonrecoverable. The error must be corrected and the loading process reinitialized.

- | | |
|----|--|
| CK | Checksum error. Backspace the input tape one record and press RUN for another attempt. |
| AR | Area reference. An attempt has been made to load a value to an area not yet defined. |
| CE | Compiler error. A terminating error occurred at compile time. |

CS Common size. A secondary use of blank common has occurred that is larger than the initially defined area.

SZ Program size. The program being loaded is too large for the memory available.

12.8.15 EXECUTION OF FORTRAN PROGRAMS

All FORTRAN main programs are loaded and entered at location 300(8). Required subprograms are loaded as they are input in successive blocks of memory. Common storage normally overlays the FORTRAN loader, which leaves the AID II routines and absolute binary loader in memory at their standard locations. Locations 0 through 77(8) are unused and locations 200(8) through 377(8) contain program and data pointers used by the program and subprograms to be executed.

To execute a FORTRAN program initialize the input/output devices selected, clear the console registers, set the program counter to 500(8), press SYSTEM RESET and RUN.

12.8.16 PROGRAMMED HALTS

DATA 620/i FORTRAN provides three types of programmed halts: STOP, PAUSE and EXIT. STOP causes the program to execute a HALT 0777 with the stop number displayed in 4-bit bcd in the A register, and the B register set to -1. A STOP implies end of job. PAUSE causes the program to execute a HALT 0000 with the pause number displayed in 4-bit bcd in the A register, and the B register set to 0. The program may be continued by pressing SYSTEM RESET and RUN. EXIT causes the program to execute a HALT 0777 with the A and B registers set to -1, and signifies end of

job. All programmed halts display error bits in the X register.

12.8.17 ERROR BIT DESIGNATIONS

Bit 0 indicates floating point overflow.
Bit 1 indicates divide check.
Bit 2 indicates fixed point overflow.
Bit 3 indicates indeterminate function.
Bit 4 indicates a log error.
Bit 5 indicates square root error.
Bit 6 indicates GO TO error.

12.8.18 ERROR HALTS

The following error halts are generated by the run time input/output package. These errors cause a 4 character message to be typed on the teletype printer followed by a call to EXIT.

FRMT: Format error.
MODE: Data mode error (floating point vs. integer).
DATA: Input data field error.
UNIT: Unit not attached or not available.
TAPE: Checksum or tape parity error.

12.8.19 BINARY INPUT/OUTPUT

All binary input/output records are a fixed length of 64 words. Paper tape records are punched with a record mark and checksum (see Appendix H-4 for a detailed format). Checksum errors encountered on input will cause a TAPE error.

12.8.20 BCD INPUT/OUTPUT

Bcd records may be fixed or variable in length depending on the device, however, only the first 80 characters are processed.

Keyboard and paper tape records are variable length and are terminated by a carriage return and line feed in that order. The character ← may be used to clear the input buffer and reset to column 1. Illegal characters are ignored. For keyboard input, the teletype bell is used to notify the operator that bcd input is required.

Card records are a fixed length of 80 characters. Illegal characters are treated as question marks and cause a DATA error unless contained within a Hollerith field. The special character ← is replaced by a blank.

Magnetic tape records are a fixed length of 84 characters, and should be card images with blank padding characters. The special character ← is replaced by a blank. Illegal characters are treated as blanks.

It should be noted that the model -33 teletype paper tape punch must be turned on and off by the operator.

Line printer records are 120 characters. Printing is left-justified with unused positions set to blanks. Vertical format control must be programmed.

APPENDIX A DATA 620/i SYSTEM ORGANIZATION

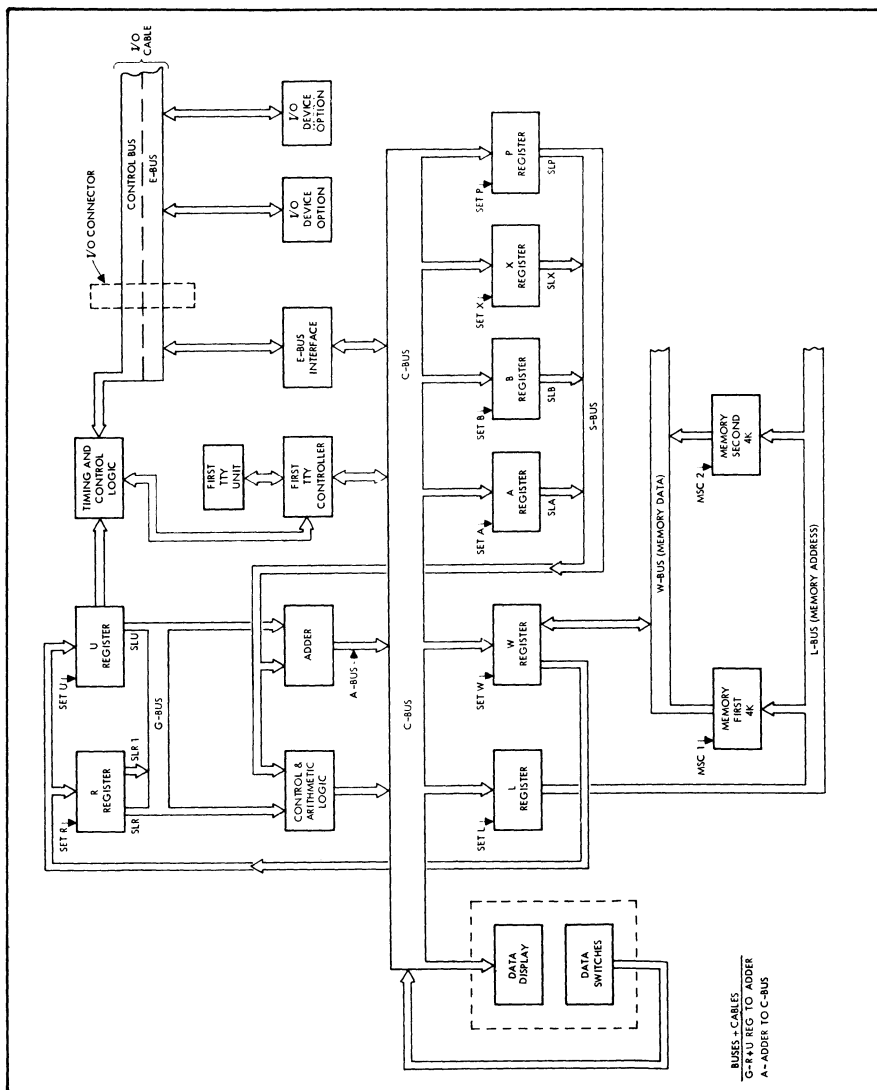


Figure A-1. DATA 620/i Internal Organization

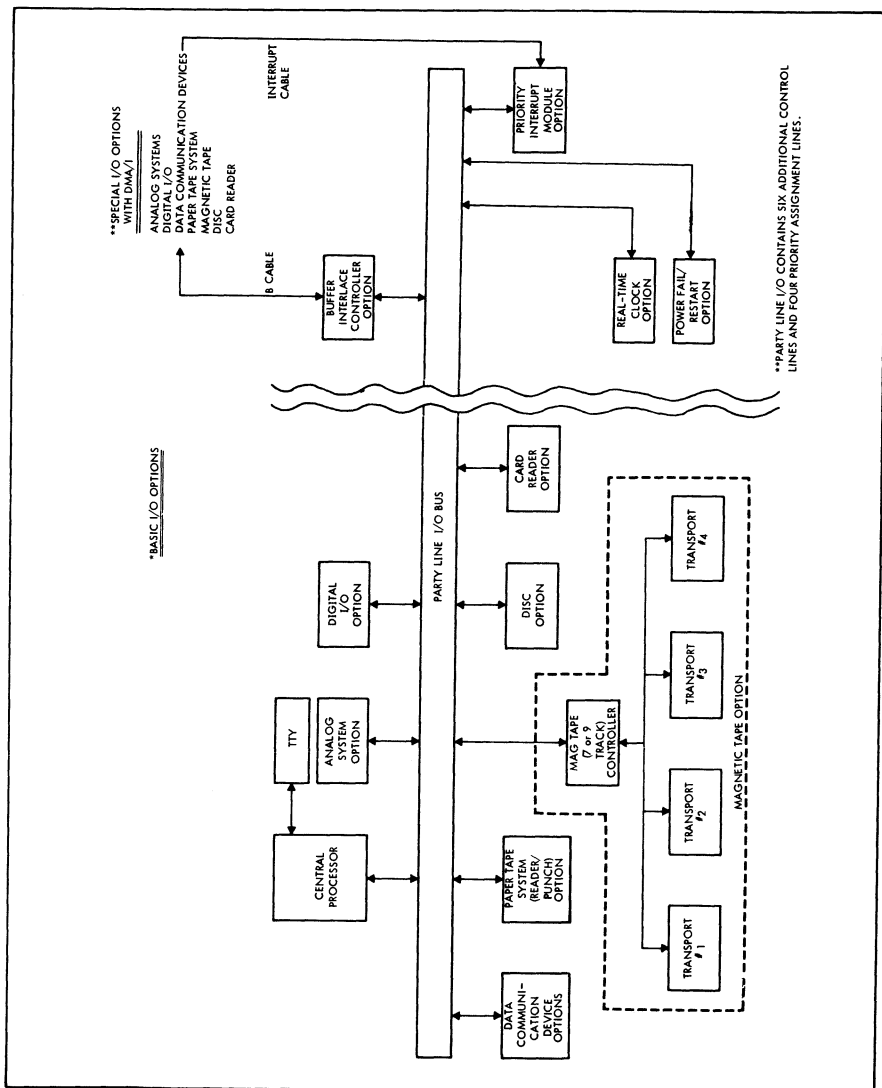


Figure A-2. DATA 620/i Input/Output Organization

APPENDIX B NUMBER SYSTEMS

B.1 DATA 620/i Number System

Binary numbers in the DATA 620/i are represented in 2's-complement form. Single-precision numbers are 15 bits plus sign (16-bit configuration) or 17 bits plus sign (18-bit configuration). The sign bit occupies the most-significant bit position (15 or 17). A "0" in the sign position denotes a positive number; a "1" in the sign position denotes a negative number. The negative of a positive number is represented in 2's-complement form.

The 2's-complement of a number may be found in either of two ways:

- a. Take the 1's-complement of the number (i.e., complement each bit); add "1" in the least-significant bit position.
Example:

+9	0000000000001001
1's-complement	1111111111110110
2's-complement (-9)	+1 1111111111110111

- b. For an n-bit number (including sign) subtract it from 2^{n+1} . Example:

2^{n+1}	1000000000000000
$- (+9)$	<u>-0000000000001001</u>
-9	1111111111110111

It is generally convenient to express binary numbers by their octal equivalent. This conversion is easily performed by grouping the binary bits by threes, starting with the least-significant bit. Thus, in the 18-bit configuration, numbers may be expressed by six full octal digits (000000-777777g).

In the 16-bit configuration, the range of octal numbers is less than six full digits (000000-177777g). The octal equivalents for the above examples are:

Decimal	Octal
+9	000011g
-9	177767g

The range of numbers in the DATA 620/i is from -2^{15} to $+2^{15}-1$ for the 16-bit configuration and -2^{17} to $+2^{17}-1$ for the 18-bit configuration. The zero minus 1 and plus/minus full-scale numbers for the 16-bit configuration are:

Binary	Octal	Decimal
0111111111111111	077777g	+32,767 +Full Scale
0000000000000000	000000	0
1111111111111111	177777g	-1
1000000000000000	100000g	-32,768 -Full Scale

The negative of the octal equivalent number is found by subtracting the number from 177777_8 and adding 1 in the least significant digit (subtract from 77777_8 for the 18-bit configuration). Example:

$$\begin{array}{r}
 177777_8 \\
 -(-9) \quad -000011_8 \\
 \hline
 +1 \\
 \hline
 (-9) \quad 177767_8
 \end{array}$$

In performing addition or subtraction, it is possible for the results to exceed the $\pm$ full scale range of the machine. For example:

Decimal	Octal
+21,980	52734 ₈
+11,843	+27103 ₈
<u>33,823</u>	<u>102037₈</u> -31,713

The negative result is in error. The same type of error occurs if the sum of the two negative numbers exceeds the minus full-scale range:

Decimal	Octal
-21,980	125044 ₈
(+)-11,843	150675 ₈
<u>-33,823</u>	<u>(1)07541₈</u> 31,803

Note that the carry out of the most-significant octal digit position is generally lost. However, to inform the programmer that the true result of an addition/subtraction falls outside

the range of the machine, an overflow indicator is provided. The overflow indicator is set if the sign bit changes when two numbers of the same sign are added together (where the sign of the subtrahend is changed in subtraction).

In multiplication, a double-length product is formed in the arithmetic registers (A or B). Since the product cannot exceed 32-bits (36-bits in the 18-bit configuration), overflow will never occur as the result of a multiply. The sign of the product is automatically determined.

Example:

Decimal	Octal
21,980	+52734
X 11,843	X +27103
<u>65,940</u>	<u>200624</u>
87,920	52734
175,840	454404
21,980	125670
21,980	<u> </u>
<u>260,299,140</u>	<u>+17410 00224</u>
	A B

The double-length result is accumulated in the A and B registers.

In division, an overflow (underflow) can occur if the divisor is less than or equal to the dividend.

B-2. Table of Powers of Two

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125

B-3. Octal-Decimal Integer Conversion Table

		0	1	2	3	4	5	6	7
0000	0000	0000	0001	0002	0003	0004	0005	0006	0007
0010	0010	0008	0009	0010	0011	0012	0013	0014	0015
0020	0020	0016	0017	0018	0019	0020	0021	0022	0023
0030	0030	0024	0025	0026	0027	0028	0029	0030	0031
0040	0040	0032	0033	0034	0035	0036	0037	0038	0039
0050	0050	0040	0041	0042	0043	0044	0045	0046	0047
0060	0060	0048	0049	0050	0051	0052	0053	0054	0055
0070	0070	0056	0057	0058	0059	0060	0061	0062	0063
Octal Decimal									
10000	4096								
20000	8192								
30000	12288								
40000	16384								
50000	20480								
60000	24576								
70000	28672								
		0100	0064	0065	0066	0067	0068	0069	0070
		0110	0072	0073	0074	0075	0076	0077	0078
		0120	0080	0081	0082	0083	0084	0085	0086
		0130	0088	0089	0090	0091	0092	0093	0094
		0140	0096	0097	0098	0099	0100	0101	0102
		0150	0104	0105	0106	0107	0108	0109	0110
		0160	0112	0113	0114	0115	0116	0117	0118
		0170	0120	0121	0122	0123	0124	0125	0126
		0200	0128	0129	0130	0131	0132	0133	0134
		0210	0136	0137	0138	0139	0140	0141	0142
		0220	0144	0145	0146	0147	0148	0149	0150
		0230	0152	0153	0154	0155	0156	0157	0158
		0240	0160	0161	0162	0163	0164	0165	0166
		0250	0168	0169	0170	0171	0172	0173	0174
		0260	0176	0177	0178	0179	0180	0181	0182
		0270	0184	0185	0186	0187	0188	0189	0190
		0300	0192	0193	0194	0195	0196	0197	0198
		0310	0200	0201	0202	0203	0204	0205	0206
		0320	0208	0209	0210	0211	0212	0213	0214
		0330	0216	0217	0218	0219	0220	0221	0222
		0340	0224	0225	0226	0227	0228	0229	0230
		0350	0232	0233	0234	0235	0236	0237	0238
		0360	0240	0241	0242	0243	0244	0245	0246
		0370	0248	0249	0250	0251	0252	0253	0254

		0	1	2	3	4	5	6	7
0400	0256	0257	0258	0259	0260	0261	0262	0263	
0410	0264	0265	0266	0267	0268	0269	0270	0271	
0420	0272	0273	0274	0275	0276	0277	0278	0279	
0430	0280	0281	0282	0283	0284	0285	0286	0287	
0440	0288	0289	0290	0291	0292	0293	0294	0295	
0450	0296	0297	0298	0299	0300	0301	0302	0303	
0460	0304	0305	0306	0307	0308	0309	0310	0311	
0470	0312	0313	0314	0315	0316	0317	0318	0319	
		0500	0320	0321	0322	0323	0324	0325	0326
		0510	0328	0329	0330	0331	0332	0333	0334
		0520	0336	0337	0338	0339	0340	0341	0342
		0530	0344	0345	0346	0347	0348	0349	0350
		0540	0352	0353	0354	0355	0356	0357	0358
		0550	0360	0361	0362	0363	0364	0365	0366
		0560	0368	0369	0370	0371	0372	0373	0374
		0570	0376	0377	0378	0379	0380	0381	0382
		0600	0384	0385	0386	0387	0388	0389	0390
		0610	0392	0393	0394	0395	0396	0397	0398
		0620	0400	0401	0402	0403	0404	0405	0406
		0630	0408	0409	0410	0411	0412	0413	0414
		0640	0416	0417	0418	0419	0420	0421	0422
		0650	0424	0425	0426	0427	0428	0429	0430
		0660	0432	0433	0434	0435	0436	0437	0438
		0670	0440	0441	0442	0443	0444	0445	0446
		0700	0448	0449	0450	0451	0452	0453	0454
		0710	0456	0457	0458	0459	0460	0461	0462
		0720	0464	0465	0466	0467	0468	0469	0470
		0730	0472	0473	0474	0475	0476	0477	0478
		0740	0480	0481	0482	0483	0484	0485	0486
		0750	0488	0489	0490	0491	0492	0493	0494
		0760	0496	0497	0498	0499	0500	0501	0502
		0770	0504	0505	0506	0507	0508	0509	0510

		0	1	2	3	4	5	6	7
1000	0512	0513	0514	0515	0516	0517	0518	0519	
1010	0520	0521	0522	0523	0524	0525	0526	0527	
1020	0528	0529	0530	0531	0532	0533	0534	0535	
1030	0536	0537	0538	0539	0540	0541	0542	0543	
1040	0544	0545	0546	0547	0548	0549	0550	0551	
1050	0552	0553	0554	0555	0556	0557	0558	0559	
1060	0560	0561	0562	0563	0564	0565	0566	0567	
1070	0568	0569	0570	0571	0572	0573	0574	0575	
		1100	0576	0577	0578	0579	0580	0581	0582
		1110	0584	0585	0586	0587	0588	0589	0590
		1120	0592	0593	0594	0595	0596	0597	0598
		1130	0600	0601	0602	0603	0604	0605	0606
		1140	0608	0609	0610	0611	0612	0613	0614
		1150	0616	0617	0618	0619	0620	0621	0622
		1160	0624	0625	0626	0627	0628	0629	0630
		1170	0632	0633	0634	0635	0636	0637	0638
		1200	0640	0641	0642	0643	0644	0645	0646
		1210	0648	0649	0650	0651	0652	0653	0654
		1220	0656	0657	0658	0659	0660	0661	0662
		1230	0664	0665	0666	0667	0668	0669	0670
		1240	0672	0673	0674	0675	0676	0677	0678
		1250	0680	0681	0682	0683	0684	0685	0686
		1260	0688	0689	0690	0691	0692	0693	0694
		1270	0696	0697	0698	0699	0700	0701	0702
		1300	0704	0705	0706	0707	0708	0709	0710
		1310	0712	0713	0714	0715	0716	0717	0718
		1320	0720	0721	0722	0723	0724	0725	0726
		1330	0728	0729	0730	0731	0732	0733	0734
		1340	0736	0737	0738	0739	0740	0741	0742
		1350	0744	0745	0746	0747	0748	0749	0750
		1360	0752	0753	0754	0755	0756	0757	0758
		1370	0760	0761	0762	0763	0764	0765	0766

	0	1	2	3	4	5	6	7
1400	0768	0769	0770	0771	0772	0773	0774	0775
1410	0776	0777	0778	0779	0780	0781	0782	0783
1420	0784	0785	0786	0787	0788	0789	0790	0791
1430	0792	0793	0794	0795	0796	0797	0798	0799
1440	0800	0801	0802	0803	0804	0805	0806	0807
1450	0808	0809	0810	0811	0812	0813	0814	0815
1460	0816	0817	0818	0819	0820	0821	0822	0823
1470	0824	0825	0826	0827	0828	0829	0830	0831
1500	0832	0833	0834	0835	0836	0837	0838	0839
1510	0840	0841	0842	0843	0844	0845	0846	0847
1520	0848	0849	0850	0851	0852	0853	0854	0855
1530	0856	0857	0858	0859	0860	0861	0862	0863
1540	0864	0865	0866	0867	0868	0869	0870	0871
1550	0872	0873	0874	0875	0876	0877	0878	0879
1560	0880	0881	0882	0883	0884	0885	0886	0887
1570	0888	0889	0890	0891	0892	0893	0894	0895
1600	0896	0897	0898	0899	0900	0901	0902	0903
1610	0904	0905	0906	0907	0908	0909	0910	0911
1620	0912	0913	0914	0915	0916	0917	0918	0919
1630	0920	0921	0922	0923	0924	0925	0926	0927
1640	0928	0929	0930	0931	0932	0933	0934	0935
1650	0936	0937	0938	0939	0940	0941	0942	0943
1660	0944	0945	0946	0947	0948	0949	0950	0951
1670	0952	0953	0954	0955	0956	0957	0958	0959
1700	0960	0961	0962	0963	0964	0965	0966	0967
1710	0968	0969	0970	0971	0972	0973	0974	0975
1720	0976	0977	0978	0979	0980	0981	0982	0983
1730	0984	0985	0986	0987	0988	0989	0990	0991
1740	0992	0993	0994	0995	0996	0997	0998	0999
1750	1000	1001	1002	1003	1004	1005	1006	1007
1760	1008	1009	1010	1011	1012	1013	1014	1015
1770	1016	1017	1018	1019	1020	1021	1022	1023

B-3. Octal-Decimal Integer Conversion Table

	0	1	2	3	4	5	6	7
2000	1024	1025	1026	1027	1028	1029	1030	1031
2010	1032	1033	1034	1035	1036	1037	1038	1039
2020	1040	1041	1042	1043	1044	1045	1046	1047
2030	1048	1049	1050	1051	1052	1053	1054	1055
2040	1056	1057	1058	1059	1060	1061	1062	1063
2050	1064	1065	1066	1067	1068	1069	1070	1071
2060	1072	1073	1074	1075	1076	1077	1078	1079
2070	1080	1081	1082	1083	1084	1085	1086	1087
2100	1088	1089	1090	1091	1092	1093	1094	1095
2110	1096	1097	1098	1099	1100	1101	1102	1103
2120	1104	1105	1106	1107	1108	1109	1110	1111
2130	1112	1113	1114	1115	1116	1117	1118	1119
2140	1120	1121	1122	1123	1124	1125	1126	1127
2150	1128	1129	1130	1131	1132	1133	1134	1135
2160	1136	1137	1138	1139	1140	1141	1142	1143
2170	1144	1145	1146	1147	1148	1149	1150	1151
2200	1152	1153	1154	1155	1156	1157	1158	1159
2210	1160	1161	1162	1163	1164	1165	1166	1167
2220	1168	1169	1170	1171	1172	1173	1174	1175
2230	1176	1177	1178	1179	1180	1181	1182	1183
2240	1184	1185	1186	1187	1188	1189	1190	1191
2250	1192	1193	1194	1195	1196	1197	1198	1199
2260	1200	1201	1202	1203	1204	1205	1206	1207
2270	1208	1209	1210	1211	1212	1213	1214	1215
2300	1216	1217	1218	1219	1220	1221	1222	1223
2310	1224	1225	1226	1227	1228	1229	1230	1231
2320	1232	1233	1234	1235	1236	1237	1238	1239
2330	1240	1241	1242	1243	1244	1245	1246	1247
2340	1248	1249	1250	1251	1252	1253	1254	1255
2350	1256	1257	1258	1259	1260	1261	1262	1263
2360	1264	1265	1266	1267	1268	1269	1270	1271
2370	1272	1273	1274	1275	1276	1277	1278	1279

	0	1	2	3	4	5	6	7
2400	1280	1281	1282	1283	1284	1285	1286	1287
2410	1288	1289	1290	1291	1292	1293	1294	1295
2420	1296	1297	1298	1299	1300	1301	1302	1303
2430	1304	1305	1306	1307	1308	1309	1310	1311
2440	1312	1313	1314	1315	1316	1317	1318	1319
2450	1320	1321	1322	1323	1324	1325	1326	1327
2460	1328	1329	1330	1331	1332	1333	1334	1335
2470	1336	1337	1338	1339	1340	1341	1342	1343
2500	1344	1345	1346	1347	1348	1349	1350	1351
2510	1352	1353	1354	1355	1356	1357	1358	1359
2520	1360	1361	1362	1363	1364	1365	1366	1367
2530	1368	1369	1370	1371	1372	1373	1374	1375
2540	1376	1377	1378	1379	1380	1381	1382	1383
2550	1384	1385	1386	1387	1388	1389	1390	1391
2560	1392	1393	1394	1395	1396	1397	1398	1399
2570	1400	1401	1402	1403	1404	1405	1406	1407
2600	1408	1409	1410	1411	1412	1413	1414	1415
2610	1416	1417	1418	1419	1420	1421	1422	1423
2620	1424	1425	1426	1427	1428	1429	1430	1431
2630	1432	1433	1434	1435	1436	1437	1438	1439
2640	1440	1441	1442	1443	1444	1445	1446	1447
2650	1448	1449	1450	1451	1452	1453	1454	1455
2660	1456	1457	1458	1459	1460	1461	1462	1463
2670	1464	1465	1466	1467	1468	1469	1470	1471
2700	1472	1473	1474	1475	1476	1477	1478	1479
2710	1480	1481	1482	1483	1484	1485	1486	1487
2720	1488	1489	1490	1491	1492	1493	1494	1495
2730	1496	1497	1498	1499	1500	1501	1502	1503
2740	1504	1505	1506	1507	1508	1509	1510	1511
2750	1512	1513	1514	1515	1516	1517	1518	1519
2760	1520	1521	1522	1523	1524	1525	1526	1527
2770	1528	1529	1530	1531	1532	1533	1534	1535

2000 1024
 10 16
 2777 1535
 (Octal) (Decimal)

Octal Decimal
 10000 - 4096
 20000 - 8192
 30000 - 12288
 40000 - 16384
 50000 - 20480
 60000 - 24576
 70000 - 28672

	0	1	2	3	4	5	6	7
3000	1536	1537	1538	1539	1540	1541	1542	1543
3010	1544	1545	1546	1547	1548	1549	1550	1551
3020	1552	1553	1554	1555	1556	1557	1558	1559
3030	1560	1561	1562	1563	1564	1565	1566	1567
3040	1568	1569	1570	1571	1572	1573	1574	1575
3050	1576	1577	1578	1579	1580	1581	1582	1583
3060	1584	1585	1586	1587	1588	1589	1590	1591
3070	1592	1593	1594	1595	1596	1597	1598	1599
3100	1600	1601	1602	1603	1604	1605	1606	1607
3110	1608	1609	1610	1611	1612	1613	1614	1615
3120	1616	1617	1618	1619	1620	1621	1622	1623
3130	1624	1625	1626	1627	1628	1629	1630	1631
3140	1632	1633	1634	1635	1636	1637	1638	1639
3150	1640	1641	1642	1643	1644	1645	1646	1647
3160	1648	1649	1650	1651	1652	1653	1654	1655
3170	1656	1657	1658	1659	1660	1661	1662	1663
3200	1664	1665	1666	1667	1668	1669	1670	1671
3210	1672	1673	1674	1675	1676	1677	1678	1679
3220	1680	1681	1682	1683	1684	1685	1686	1687
3230	1688	1689	1690	1691	1692	1693	1694	1695
3240	1696	1697	1698	1699	1700	1701	1702	1703
3250	1704	1705	1706	1707	1708	1709	1710	1711
3260	1712	1713	1714	1715	1716	1717	1718	1719
3270	1720	1721	1722	1723	1724	1725	1726	1727
3300	1728	1729	1730	1731	1732	1733	1734	1735
3310	1736	1737	1738	1739	1740	1741	1742	1743
3320	1744	1745	1746	1747	1748	1749	1750	1751
3330	1752	1753	1754	1755	1756	1757	1758	1759
3340	1760	1761	1762	1763	1764	1765	1766	1767
3350	1768	1769	1770	1771	1772	1773	1774	1775
3360	1776	1777	1778	1779	1780	1781	1782	1783
3370	1784	1785	1786	1787	1788	1789	1790	1791

	0	1	2	3	4	5	6	7
3400	1792	1793	1794	1795	1796	1797	1798	1799
3410	1800	1801	1802	1803	1804	1805	1806	1807
3420	1808	1809	1810	1811	1812	1813	1814	1815
3430	1816	1817	1818	1819	1820	1821	1822	1823
3440	1824	1825	1826	1827	1828	1829	1830	1831
3450	1832	1833	1834	1835	1836	1837	1838	1839
3460	1840	1841	1842	1843	1844	1845	1846	1847
3470	1848	1849	1850	1851	1852	1853	1854	1855
3500	1856	1857	1858	1859	1860	1861	1862	1863
3510	1864	1865	1866	1867	1868	1869	1870	1871
3520	1872	1873	1874	1875	1876	1877	1878	1879
3530	1880	1881	1882	1883	1884	1885	1886	1887
3540	1888	1889	1890	1891	1892	1893	1894	1895
3550	1896	1897	1898	1899	1900	1901	1902	1903
3560	1904	1905	1906	1907	1908	1909	1910	1911
3570	1912	1913	1914	1915	1916	1917	1918	1919
3600	1920	1921	1922	1923	1924	1925	1926	1927
3610	1928	1929	1930	1931	1932	1933	1934	1935
3620	1936	1937	1938	1939	1940	1941	1942	1943
3630	1944	1945	1946	1947	1948	1949	1950	1951
3640	1952	1953	1954	1955	1956	1957	1958	1959
3650	1960	1961	1962	1963	1964	1965	1966	1967
3660	1968	1969	1970	1971	1972	1973	1974	1975
3670	1976	1977	1978	1979	1980	1981	1982	1983
3700	1984	1985	1986	1987	1988	1989	1990	1991
3710	1992	1993	1994	1995	1996	1997	1998	1999
3720	2000	2001	2002	2003	2004	2005	2006	2007
3730	2008	2009	2010	2011	2012	2013	2014	2015
3740	2016	2017	2018	2019	2020	2021	2022	2023
3750	2024	2025	2026	2027	2028	2029	2030	2031
3760	2032	2033	2034	2035	2036	2037	2038	2039
3770	2040	2041	2042	2043	2044	2045	2046	2047

3000 1536
 10 16
 3777 2047
 (Octal) (Decimal)

B-3. Octal-Decimal Integer Conversion Table

		0	1	2	3	4	5	6	7			0	1	2	3	4	5	6	7
4000	7048	4000	2048	2049	2050	2051	2052	2053	2054	2055	4400	2304	2305	2306	2307	2308	2309	2310	2311
	to	4010	2056	2057	2058	2059	2060	2061	2062	2063	4410	2312	2313	2314	2315	2316	2317	2318	2319
4777	2559	4020	2064	2065	2066	2067	2068	2069	2070	2071	4420	2320	2321	2322	2323	2324	2325	2326	2327
(Octal)	(Decimal)	4030	2072	2073	2074	2075	2076	2077	2078	2079	4430	2328	2329	2330	2331	2332	2333	2334	2335
Octal	Decimal	4040	2080	2081	2082	2083	2084	2085	2086	2087	4440	2336	2337	2338	2339	2340	2341	2342	2343
10000	4096	4050	2088	2089	2090	2091	2092	2093	2094	2095	4450	2344	2345	2346	2347	2348	2349	2350	2351
20000	8192	4060	2096	2097	2098	2099	2100	2101	2102	2103	4460	2352	2353	2354	2355	2356	2357	2358	2359
30000	12288	4070	2104	2105	2106	2107	2108	2109	2110	2111	4470	2360	2361	2362	2363	2364	2365	2366	2367
40000	16384	4100	2112	2113	2114	2115	2116	2117	2118	2119	4500	2368	2369	2370	2371	2372	2373	2374	2375
50000	20480	4110	2120	2121	2122	2123	2124	2125	2126	2127	4510	2376	2377	2378	2379	2380	2381	2382	2383
60000	24576	4120	2128	2129	2130	2131	2132	2133	2134	2135	4520	2384	2385	2386	2387	2388	2389	2390	2391
70000	28672	4130	2136	2137	2138	2139	2140	2141	2142	2143	4530	2392	2393	2394	2395	2396	2397	2398	2399
		4140	2144	2145	2146	2147	2148	2149	2150	2151	4540	2400	2401	2402	2403	2404	2405	2406	2407
		4150	2152	2153	2154	2155	2156	2157	2158	2159	4550	2408	2409	2410	2411	2412	2413	2414	2415
		4160	2160	2161	2162	2163	2164	2165	2166	2167	4560	2416	2417	2418	2419	2420	2421	2422	2423
		4170	2168	2169	2170	2171	2172	2173	2174	2175	4570	2424	2425	2426	2427	2428	2429	2430	2431
		4200	2176	2177	2178	2179	2180	2181	2182	2183	4600	2432	2433	2434	2435	2436	2437	2438	2439
		4210	2184	2185	2186	2187	2188	2189	2190	2191	4610	2440	2441	2442	2443	2444	2445	2446	2447
		4220	2192	2193	2194	2195	2196	2197	2198	2199	4620	2448	2449	2450	2451	2452	2453	2454	2455
		4230	2200	2201	2202	2203	2204	2205	2206	2207	4630	2456	2457	2458	2459	2460	2461	2462	2463
		4240	2208	2209	2210	2211	2212	2213	2214	2215	4640	2464	2465	2466	2467	2468	2469	2470	2471
		4250	2216	2217	2218	2219	2220	2221	2222	2223	4650	2472	2473	2474	2475	2476	2477	2478	2479
		4260	2224	2225	2226	2227	2228	2229	2230	2231	4660	2480	2481	2482	2483	2484	2485	2486	2487
		4270	2232	2233	2234	2235	2236	2237	2238	2239	4670	2488	2489	2490	2491	2492	2493	2494	2495
		4300	2240	2241	2242	2243	2244	2245	2246	2247	4700	2496	2497	2498	2499	2500	2501	2502	2503
		4310	2248	2249	2250	2251	2252	2253	2254	2255	4710	2504	2505	2506	2507	2508	2509	2510	2511
		4320	2256	2257	2258	2259	2260	2261	2262	2263	4720	2512	2513	2514	2515	2516	2517	2518	2519
		4330	2264	2265	2266	2267	2268	2269	2270	2271	4730	2520	2521	2522	2523	2524	2525	2526	2527
		4340	2272	2273	2274	2275	2276	2277	2278	2279	4740	2528	2529	2530	2531	2532	2533	2534	2535
		4350	2280	2281	2282	2283	2284	2285	2286	2287	4750	2536	2537	2538	2539	2540	2541	2542	2543
		4360	2288	2289	2290	2291	2292	2293	2294	2295	4760	2544	2545	2546	2547	2548	2549	2550	2551
		4370	2296	2297	2298	2299	2300	2301	2302	2303	4770	2552	2553	2554	2555	2556	2557	2558	2559

		0	1	2	3	4	5	6	7			0	1	2	3	4	5	6	7
5000	2560	5000	2561	2562	2563	2564	2565	2566	2567	5400	2816	2817	2818	2819	2820	2821	2822	2823	
	to	5010	2568	2569	2570	2571	2572	2573	2574	5410	2824	2825	2826	2827	2828	2829	2830	2831	
5777	3071	5020	2576	2577	2578	2579	2580	2581	2582	5420	2832	2833	2834	2835	2836	2837	2838	2839	
(Octal)	(Decimal)	5030	2584	2585	2586	2587	2588	2589	2590	5430	2840	2841	2842	2843	2844	2845	2846	2847	
		5040	2592	2593	2594	2595	2596	2597	2598	5440	2848	2849	2850	2851	2852	2853	2854	2855	
		5050	2600	2601	2602	2603	2604	2605	2606	5450	2856	2857	2858	2859	2860	2861	2862	2863	
		5060	2608	2609	2610	2611	2612	2613	2614	5460	2864	2865	2866	2867	2868	2869	2870	2871	
		5070	2616	2617	2618	2619	2620	2621	2622	5470	2872	2873	2874	2875	2876	2877	2878	2879	
		5100	2624	2625	2626	2627	2628	2629	2630	5500	2880	2881	2882	2883	2884	2885	2886	2887	
		5110	2632	2633	2634	2635	2636	2637	2638	5510	2888	2889	2890	2891	2892	2893	2894	2895	
		5120	2640	2641	2642	2643	2644	2645	2646	5520	2896	2897	2898	2899	2900	2901	2902	2903	
		5130	2648	2649	2650	2651	2652	2653	2654	5530	2904	2905	2906	2907	2908	2909	2910	2911	
		5140	2656	2657	2658	2659	2660	2661	2662	5540	2912	2913	2914	2915	2916	2917	2918	2919	
		5150	2664	2665	2666	2667	2668	2669	2670	5550	2920	2921	2922	2923	2924	2925	2926	2927	
		5160	2672	2673	2674	2675	2676	2677	2678	5560	2928	2929	2930	2931	2932	2933	2934	2935	
		5170	2680	2681	2682	2683	2684	2685	2686	5570	2936	2937	2938	2939	2940	2941	2942	2943	
		5200	2688	2689	2690	2691	2692	2693	2694	5600	2944	2945	2946	2947	2948	2949	2950	2951	
		5210	2696	2697	2698	2699	2700	2701	2702	5610	2952	2953	2954	2955	2956	2957	2958	2959	
		5220	2704	2705	2706	2707	2708	2709	2710	5620	2960	2961	2962	2963	2964	2965	2966	2967	
		5230	2712	2713	2714	2715	2716	2717	2718	5630	2968	2969	2970	2971	2972	2973	2974	2975	
		5240	2720	2721	2722	2723	2724	2725	2726	5640	2976	2977	2978	2979	2980	2981	2982	2983	
		5250	2728	2729	2730	2731	2732	2733	2734	5650	2984	2985	2986	2987	2988	2989	2990	2991	
		5260	2736	2737	2738	2739	2740	2741	2742	5660	2992	2993	2994	2995	2996	2997	2998	2999	
		5270	2744	2745	2746	2747	2748	2749	2750	5670	3000	3001	3002	3003	3004	3005	3006	3007	
		5300	2752	2753	2754	2755	2756	2757	2758	5700	3008	3009	3010	3011	3012	3013	3014	3015	
		5310	2760	2761	2762	2763	2764	2765	2766	5710	3016	3017	3018	3019	3020	3021	3022	3023	
		5320	2768	2769	2770	2771	2772	2773	2774	5720	3024	3025	3026	3027	3028	3029	3030	3031	
		5330	2776	2777	2778	2779	2780	2781	2782	5730	3032	3033	3034	3035	3036	3037	3038	3039	
		5340	2784	2785	2786	2787	2788	2789	2790	5740	3040	3041	3042	3043	3044	3045	3046	3047	
		5350	2792	2793	2794	2795	2796	2797	2798	5750	3048	3049	3050	3051	3052	3053	3054	3055	
		5360	2800	2801	2802	2803	2804	2805	2806	5760	3056	3057	3058	3059	3060	3061	3062	3063	
		5370	2808	2809	2810	2811	2812	2813	2814	5770	3064	3065	3066	3067	3068	3069	3070	3071	

B-3. Octal-Decimal Integer Conversion Table

	0	1	2	3	4	5	6	7
6000	3072	3073	3074	3075	3076	3077	3078	3079
6010	3080	3081	3082	3083	3084	3085	3086	3087
6020	3088	3089	3090	3091	3092	3093	3094	3095
6030	3096	3097	3098	3099	3100	3101	3102	3103
6040	3104	3105	3106	3107	3108	3109	3110	3111
6050	3112	3113	3114	3115	3116	3117	3118	3119
6060	3120	3121	3122	3123	3124	3125	3126	3127
6070	3128	3129	3130	3131	3132	3133	3134	3135
6100	3136	3137	3138	3139	3140	3141	3142	3143
6110	3144	3145	3146	3147	3148	3149	3150	3151
6120	3152	3153	3154	3155	3156	3157	3158	3159
6130	3160	3161	3162	3163	3164	3165	3166	3167
6140	3168	3169	3170	3171	3172	3173	3174	3175
6150	3176	3177	3178	3179	3180	3181	3182	3183
6160	3184	3185	3186	3187	3188	3189	3190	3191
6170	3192	3193	3194	3195	3196	3197	3198	3199
6200	3200	3201	3202	3203	3204	3205	3206	3207
6210	3208	3209	3210	3211	3212	3213	3214	3215
6220	3216	3217	3218	3219	3220	3221	3222	3223
6230	3224	3225	3226	3227	3228	3229	3230	3231
6240	3232	3233	3234	3235	3236	3237	3238	3239
6250	3240	3241	3242	3243	3244	3245	3246	3247
6260	3248	3249	3250	3251	3252	3253	3254	3255
6270	3256	3257	3258	3259	3260	3261	3262	3263
6300	3264	3265	3266	3267	3268	3269	3270	3271
6310	3272	3273	3274	3275	3276	3277	3278	3279
6320	3280	3281	3282	3283	3284	3285	3286	3287
6330	3288	3289	3290	3291	3292	3293	3294	3295
6340	3296	3297	3298	3299	3300	3301	3302	3303
6350	3304	3305	3306	3307	3308	3309	3310	3311
6360	3312	3313	3314	3315	3316	3317	3318	3319
6370	3320	3321	3322	3323	3324	3325	3326	3327

	0	1	2	3	4	5	6	7
6400	3328	3329	3330	3331	3332	3333	3334	3335
6410	3336	3337	3338	3339	3340	3341	3342	3343
6420	3344	3345	3346	3347	3348	3349	3350	3351
6430	3352	3353	3354	3355	3356	3357	3358	3359
6440	3360	3361	3362	3363	3364	3365	3366	3367
6450	3368	3369	3370	3371	3372	3373	3374	3375
6460	3376	3377	3378	3379	3380	3381	3382	3383
6470	3384	3385	3386	3387	3388	3389	3390	3391
6500	3392	3393	3394	3395	3396	3397	3398	3399
6510	3400	3401	3402	3403	3404	3405	3406	3407
6520	3408	3409	3410	3411	3412	3413	3414	3415
6530	3416	3417	3418	3419	3420	3421	3422	3423
6540	3424	3425	3426	3427	3428	3429	3430	3431
6550	3432	3433	3434	3435	3436	3437	3438	3439
6560	3440	3441	3442	3443	3444	3445	3446	3447
6570	3448	3449	3450	3451	3452	3453	3454	3455
6600	3456	3457	3458	3459	3460	3461	3462	3463
6610	3464	3465	3466	3467	3468	3469	3470	3471
6620	3472	3473	3474	3475	3476	3477	3478	3479
6630	3480	3481	3482	3483	3484	3485	3486	3487
6640	3488	3489	3490	3491	3492	3493	3494	3495
6650	3496	3497	3498	3499	3500	3501	3502	3503
6660	3504	3505	3506	3507	3508	3509	3510	3511
6670	3512	3513	3514	3515	3516	3517	3518	3519
6700	3520	3521	3522	3523	3524	3525	3526	3527
6710	3528	3529	3530	3531	3532	3533	3534	3535
6720	3536	3537	3538	3539	3540	3541	3542	3543
6730	3544	3545	3546	3547	3548	3549	3550	3551
6740	3552	3553	3554	3555	3556	3557	3558	3559
6750	3560	3561	3562	3563	3564	3565	3566	3567
6760	3568	3569	3570	3571	3572	3573	3574	3575
6770	3576	3577	3578	3579	3580	3581	3582	3583

6000 to 6077
(Octal) (Decimal)

Octal Decimal
10000 - 4096
20000 - 8192
30000 - 12288
40000 - 16384
50000 - 20480
60000 - 24576
70000 - 28672

	0	1	2	3	4	5	6	7
7000	3584	3585	3586	3587	3588	3589	3590	3591
7010	3592	3593	3594	3595	3596	3597	3598	3599
7020	3600	3601	3602	3603	3604	3605	3606	3607
7030	3608	3609	3610	3611	3612	3613	3614	3615
7040	3616	3617	3618	3619	3620	3621	3622	3623
7050	3624	3625	3626	3627	3628	3629	3630	3631
7060	3632	3633	3634	3635	3636	3637	3638	3639
7070	3640	3641	3642	3643	3644	3645	3646	3647
7100	3648	3649	3650	3651	3652	3653	3654	3655
7110	3656	3657	3658	3659	3660	3661	3662	3663
7120	3664	3665	3666	3667	3668	3669	3670	3671
7130	3672	3673	3674	3675	3676	3677	3678	3679
7140	3680	3681	3682	3683	3684	3685	3686	3687
7150	3688	3689	3690	3691	3692	3693	3694	3695
7160	3696	3697	3698	3699	3700	3701	3702	3703
7170	3704	3705	3706	3707	3708	3709	3710	3711
7200	3712	3713	3714	3715	3716	3717	3718	3719
7210	3720	3721	3722	3723	3724	3725	3726	3727
7220	3728	3729	3730	3731	3732	3733	3734	3735
7230	3736	3737	3738	3739	3740	3741	3742	3743
7240	3744	3745	3746	3747	3748	3749	3750	3751
7250	3752	3753	3754	3755	3756	3757	3758	3759
7260	3760	3761	3762	3763	3764	3765	3766	3767
7270	3768	3769	3770	3771	3772	3773	3774	3775
7300	3776	3777	3778	3779	3780	3781	3782	3783
7310	3784	3785	3786	3787	3788	3789	3790	3791
7320	3792	3793	3794	3795	3796	3797	3798	3799
7330	3800	3801	3802	3803	3804	3805	3806	3807
7340	3808	3809	3810	3811	3812	3813	3814	3815
7350	3816	3817	3818	3819	3820	3821	3822	3823
7360	3824	3825	3826	3827	3828	3829	3830	3831
7370	3832	3833	3834	3835	3836	3837	3838	3839

	0	1	2	3	4	5	6	7
7400	3840	3841	3842	3843	3844	3845	3846	3847
7410	3848	3849	3850	3851	3852	3853	3854	3855
7420	3856	3857	3858	3859	3860	3861	3862	3863
7430	3864	3865	3866	3867	3868	3869	3870	3871
7440	3872	3873	3874	3875	3876	3877	3878	3879
7450	3880	3881	3882	3883	3884	3885	3886	3887
7460	3888	3889	3890	3891	3892	3893	3894	3895
7470	3896	3897	3898	3899	3900	3901	3902	3903
7500	3904	3905	3906	3907	3908	3909	3910	3911
7510	3912	3913	3914	3915	3916	3917	3918	3919
7520	3920	3921	3922	3923	3924	3925	3926	3927
7530	3928	3929	3930	3931	3932	3933	3934	3935
7540	3936	3937	3938	3939	3940	3941	3942	3943
7550	3944	3945	3946	3947	3948	3949	3950	3951
7560	3952	3953	3954	3955	3956	3957	3958	3959
7570	3960	3961	3962	3963	3964	3965	3966	3967
7600	3968	3969	3970	3971	3972	3973	3974	3975
7610	3976	3977	3978	3979	3980	3981	3982	3983
7620	3984	3985	3986	3987	3988	3989	3990	3991
7630	3992	3993	3994	3995	3996	3997	3998	3999
7640	4000	4001	4002	4003	4004	4005	4006	4007
7650	4008	4009	4010	4011	4012	4013	4014	4015
7660	4016	4017	4018	4019	4020	4021	4022	4023
7670	4024	4025	4026	4027	4028	4029	4030	4031
7700	4032	4033	4034	4035	4036	4037	4038	4039
7710	4040	4041	4042	4043	4044	4045	4046	4047
7720	4048	4049	4050	4051	4052	4053	4054	4055
7730	4056	4057	4058	4059	4060	4061	4062	4063
7740	4064	4065	4066	4067	4068	4069	4070	4071
7750	4072	4073	4074	4075	4076	4077	4078	4079
7760	4080	4081	4082	4083	4084	4085	4086	4087
7770	4088	4089	4090	4091	4092	4093	4094	4095

7000 to 7077
(Octal) (Decimal)

B-4. Octal-Decimal Fraction Conversion Table

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000	.000000	.100	.125000	.200	.250000	.300	.375000
.001	.001953	.101	.126953	.201	.251953	.301	.376953
.002	.003906	.102	.128906	.202	.253906	.302	.378906
.003	.005859	.103	.130859	.203	.255859	.303	.380859
.004	.007812	.104	.132812	.204	.257812	.304	.382812
.005	.009765	.105	.134765	.205	.259765	.305	.384765
.006	.011718	.106	.136718	.206	.261718	.306	.386718
.007	.013671	.107	.138671	.207	.263671	.307	.388671
.010	.015625	.110	.140625	.210	.265625	.310	.390625
.011	.017578	.111	.142578	.211	.267578	.311	.392578
.012	.019531	.112	.144531	.212	.269531	.312	.394531
.013	.021484	.113	.146484	.213	.271484	.313	.396484
.014	.023437	.114	.148437	.214	.273437	.314	.398437
.015	.025390	.115	.150390	.215	.275390	.315	.400390
.016	.027343	.116	.152343	.216	.277343	.316	.402343
.017	.029296	.117	.154296	.217	.279296	.317	.404296
.020	.031250	.120	.156250	.220	.281250	.320	.406250
.021	.033203	.121	.158203	.221	.283203	.321	.408203
.022	.035156	.122	.160156	.222	.285156	.322	.410156
.023	.037109	.123	.162109	.223	.287109	.323	.412109
.024	.039062	.124	.164062	.224	.289062	.324	.414062
.025	.041015	.125	.166015	.225	.291015	.325	.416015
.026	.042968	.126	.167968	.226	.292968	.326	.417968
.027	.044921	.127	.169921	.227	.294921	.327	.419921
.030	.046875	.130	.171875	.230	.296875	.330	.421875
.031	.048828	.131	.173828	.231	.298828	.331	.423828
.032	.050781	.132	.175781	.232	.300781	.332	.425781
.033	.052734	.133	.177734	.233	.302734	.333	.427734
.034	.054687	.134	.179687	.234	.304687	.334	.429687
.035	.056640	.135	.181640	.235	.306640	.335	.431640
.036	.058593	.136	.183593	.236	.308593	.336	.433593
.037	.060546	.137	.185546	.237	.310546	.337	.435546
.040	.062500	.140	.187500	.240	.312500	.340	.437500
.041	.064453	.141	.189453	.241	.314453	.341	.439453
.042	.066406	.142	.191406	.242	.316406	.342	.441406
.043	.068359	.143	.193359	.243	.318359	.343	.443359
.044	.070312	.144	.195312	.244	.320312	.344	.445312
.045	.072265	.145	.197265	.245	.322265	.345	.447265
.046	.074218	.146	.199218	.246	.324218	.346	.449218
.047	.076171	.147	.201171	.247	.326171	.347	.451171
.050	.078125	.150	.203125	.250	.328125	.350	.453125
.051	.080078	.151	.205078	.251	.330078	.351	.455078
.052	.082031	.152	.207031	.252	.332031	.352	.457031
.053	.083984	.153	.208984	.253	.333984	.353	.458984
.054	.085937	.154	.210937	.254	.335937	.354	.460937
.055	.087890	.155	.212890	.255	.337890	.355	.462890
.056	.089843	.156	.214843	.256	.339843	.356	.464843
.057	.091796	.157	.216796	.257	.341796	.357	.466796
.060	.093750	.160	.218750	.260	.343750	.360	.468750
.061	.095703	.161	.220703	.261	.345703	.361	.470703
.062	.097656	.162	.222656	.262	.347656	.362	.472656
.063	.099609	.163	.224609	.263	.349609	.363	.474609
.064	.101562	.164	.226562	.264	.351562	.364	.476562
.065	.103515	.165	.228515	.265	.353515	.365	.478515
.066	.105468	.166	.230468	.266	.355468	.366	.480468
.067	.107421	.167	.232421	.267	.357421	.367	.482421
.070	.109375	.170	.234375	.270	.359375	.370	.484375
.071	.111328	.171	.236328	.271	.361328	.371	.486328
.072	.113281	.172	.238281	.272	.363281	.372	.488281
.073	.115234	.173	.240234	.273	.365234	.373	.490234
.074	.117187	.174	.242187	.274	.367187	.374	.492187
.075	.119140	.175	.244140	.275	.369140	.375	.494140
.076	.121093	.176	.246093	.276	.371093	.376	.496093
.077	.123046	.177	.248046	.277	.373046	.377	.498046

B-4. Octal-Decimal Fraction Conversion Table

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000000	.000000	.000100	.000244	.000200	.000488	.000300	.000732
.000001	.000003	.000101	.000247	.000201	.000492	.000301	.000736
.000002	.000007	.000102	.000251	.000202	.000495	.000302	.000740
.000003	.000011	.000103	.000255	.000203	.000499	.000303	.000743
.000004	.000015	.000104	.000259	.000204	.000503	.000304	.000747
.000005	.000019	.000105	.000263	.000205	.000507	.000305	.000751
.000006	.000022	.000106	.000267	.000206	.000511	.000306	.000755
.000007	.000026	.000107	.000270	.000207	.000514	.000307	.000759
.000010	.000030	.000110	.000274	.000210	.000518	.000310	.000762
.000011	.000034	.000111	.000278	.000211	.000522	.000311	.000766
.000012	.000038	.000112	.000282	.000212	.000526	.000312	.000770
.000013	.000041	.000113	.000286	.000213	.000530	.000313	.000774
.000014	.000045	.000114	.000289	.000214	.000534	.000314	.000778
.000015	.000049	.000115	.000293	.000215	.000537	.000315	.000782
.000016	.000053	.000116	.000297	.000216	.000541	.000316	.000785
.000017	.000057	.000117	.000301	.000217	.000545	.000317	.000789
.000020	.000061	.000120	.000305	.000220	.000549	.000320	.000793
.000021	.000064	.000121	.000308	.000221	.000553	.000321	.000797
.000022	.000068	.000122	.000312	.000222	.000556	.000322	.000801
.000023	.000072	.000123	.000316	.000223	.000560	.000323	.000805
.000024	.000076	.000124	.000320	.000224	.000564	.000324	.000808
.000025	.000080	.000125	.000324	.000225	.000568	.000325	.000812
.000026	.000083	.000126	.000328	.000226	.000572	.000326	.000816
.000027	.000087	.000127	.000331	.000227	.000576	.000327	.000820
.000030	.000091	.000130	.000335	.000230	.000579	.000330	.000823
.000031	.000095	.000131	.000339	.000231	.000583	.000331	.000827
.000032	.000099	.000132	.000343	.000232	.000587	.000332	.000831
.000033	.000102	.000133	.000347	.000233	.000591	.000333	.000835
.000034	.000106	.000134	.000350	.000234	.000595	.000334	.000839
.000035	.000110	.000135	.000354	.000235	.000598	.000335	.000843
.000036	.000114	.000136	.000358	.000236	.000602	.000336	.000846
.000037	.000118	.000137	.000362	.000237	.000606	.000337	.000850
.000040	.000122	.000140	.000366	.000240	.000610	.000340	.000854
.000041	.000125	.000141	.000370	.000241	.000614	.000341	.000858
.000042	.000129	.000142	.000373	.000242	.000617	.000342	.000862
.000043	.000133	.000143	.000377	.000243	.000621	.000343	.000865
.000044	.000137	.000144	.000381	.000244	.000625	.000344	.000869
.000045	.000141	.000145	.000385	.000245	.000629	.000345	.000873
.000046	.000144	.000146	.000389	.000246	.000633	.000346	.000877
.000047	.000148	.000147	.000392	.000247	.000637	.000347	.000881
.000050	.000152	.000150	.000396	.000250	.000640	.000350	.000885
.000051	.000156	.000151	.000400	.000251	.000644	.000351	.000888
.000052	.000160	.000152	.000404	.000252	.000648	.000352	.000892
.000053	.000164	.000153	.000408	.000253	.000652	.000353	.000896
.000054	.000167	.000154	.000411	.000254	.000656	.000354	.000900
.000055	.000171	.000155	.000415	.000255	.000659	.000355	.000904
.000056	.000175	.000156	.000419	.000256	.000663	.000356	.000907
.000057	.000179	.000157	.000423	.000257	.000667	.000357	.000911
.000060	.000183	.000160	.000427	.000260	.000671	.000360	.000915
.000061	.000186	.000161	.000431	.000261	.000675	.000361	.000919
.000062	.000190	.000162	.000434	.000262	.000679	.000362	.000923
.000063	.000194	.000163	.000438	.000263	.000682	.000363	.000926
.000064	.000198	.000164	.000442	.000264	.000686	.000364	.000930
.000065	.000202	.000165	.000446	.000265	.000690	.000365	.000934
.000066	.000205	.000166	.000450	.000266	.000694	.000366	.000938
.000067	.000209	.000167	.000453	.000267	.000698	.000367	.000942
.000070	.000213	.000170	.000457	.000270	.000701	.000370	.000946
.000071	.000217	.000171	.000461	.000271	.000705	.000371	.000949
.000072	.000221	.000172	.000465	.000272	.000709	.000372	.000953
.000073	.000225	.000173	.000469	.000273	.000713	.000373	.000957
.000074	.000228	.000174	.000473	.000274	.000717	.000374	.000961
.000075	.000232	.000175	.000476	.000275	.000720	.000375	.000965
.000076	.000236	.000176	.000480	.000276	.000724	.000376	.000968
.000077	.000240	.000177	.000484	.000277	.000728	.000377	.000972

B-4. Octal-Decimal Fraction Conversion Table

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000400	.000976	.000500	.001220	.000600	.001464	.000700	.001708
.000401	.000980	.000501	.001224	.000601	.001468	.000701	.001712
.000402	.000984	.000502	.001228	.000602	.001472	.000702	.001716
.000403	.000988	.000503	.001232	.000603	.001476	.000703	.001720
.000404	.000991	.000504	.001235	.000604	.001480	.000704	.001724
.000405	.000995	.000505	.001239	.000605	.001483	.000705	.001728
.000406	.000999	.000506	.001243	.000606	.001487	.000706	.001731
.000407	.001003	.000507	.001247	.000607	.001491	.000707	.001735
.000410	.001007	.000510	.001251	.000610	.001495	.000710	.001739
.000411	.001010	.000511	.001255	.000611	.001499	.000711	.001743
.000412	.001014	.000512	.001258	.000612	.001502	.000712	.001747
.000413	.001018	.000513	.001262	.000613	.001506	.000713	.001750
.000414	.001022	.000514	.001266	.000614	.001510	.000714	.001754
.000415	.001026	.000515	.001270	.000615	.001514	.000715	.001758
.000416	.001029	.000516	.001274	.000616	.001518	.000716	.001762
.000417	.001033	.000517	.001277	.000617	.001522	.000717	.001766
.000420	.001037	.000520	.001281	.000620	.001525	.000720	.001770
.000421	.001041	.000521	.001285	.000621	.001529	.000721	.001773
.000422	.001045	.000522	.001289	.000622	.001533	.000722	.001777
.000423	.001049	.000523	.001293	.000623	.001537	.000723	.001781
.000424	.001052	.000524	.001296	.000624	.001541	.000724	.001785
.000425	.001056	.000525	.001300	.000625	.001544	.000725	.001789
.000426	.001060	.000526	.001304	.000626	.001548	.000726	.001792
.000427	.001064	.000527	.001308	.000627	.001552	.000727	.001796
.000430	.001068	.000530	.001312	.000630	.001556	.000730	.001800
.000431	.001071	.000531	.001316	.000631	.001560	.000731	.001804
.000432	.001075	.000532	.001319	.000632	.001564	.000732	.001808
.000433	.001079	.000533	.001323	.000633	.001567	.000733	.001811
.000434	.001083	.000534	.001327	.000634	.001571	.000734	.001815
.000435	.001087	.000535	.001331	.000635	.001575	.000735	.001819
.000436	.001091	.000536	.001335	.000636	.001579	.000736	.001823
.000437	.001094	.000537	.001338	.000637	.001583	.000737	.001827
.000440	.001098	.000540	.001342	.000640	.001586	.000740	.001831
.000441	.001102	.000541	.001346	.000641	.001590	.000741	.001834
.000442	.001106	.000542	.001350	.000642	.001594	.000742	.001838
.000443	.001110	.000543	.001354	.000643	.001598	.000743	.001842
.000444	.001113	.000544	.001358	.000644	.001602	.000744	.001846
.000445	.001117	.000545	.001361	.000645	.001605	.000745	.001850
.000446	.001121	.000546	.001365	.000646	.001609	.000746	.001853
.000447	.001125	.000547	.001369	.000647	.001613	.000747	.001857
.000450	.001129	.000550	.001373	.000650	.001617	.000750	.001861
.000451	.001132	.000551	.001377	.000651	.001621	.000751	.001865
.000452	.001136	.000552	.001380	.000652	.001625	.000752	.001869
.000453	.001140	.000553	.001384	.000653	.001628	.000753	.001873
.000454	.001144	.000554	.001388	.000654	.001632	.000754	.001876
.000455	.001148	.000555	.001392	.000655	.001636	.000755	.001880
.000456	.001152	.000556	.001396	.000656	.001640	.000756	.001884
.000457	.001155	.000557	.001399	.000657	.001644	.000757	.001888
.000460	.001159	.000560	.001403	.000660	.001647	.000760	.001892
.000461	.001163	.000561	.001407	.000661	.001651	.000761	.001895
.000462	.001167	.000562	.001411	.000662	.001655	.000762	.001899
.000463	.001171	.000563	.001415	.000663	.001659	.000763	.001903
.000464	.001174	.000564	.001419	.000664	.001663	.000764	.001907
.000465	.001178	.000565	.001422	.000665	.001667	.000765	.001911
.000466	.001182	.000566	.001426	.000666	.001670	.000766	.001914
.000467	.001186	.000567	.001430	.000667	.001674	.000767	.001918
.000470	.001190	.000570	.001434	.000670	.001678	.000770	.001922
.000471	.001194	.000571	.001438	.000671	.001682	.000771	.001926
.000472	.001197	.000572	.001441	.000672	.001686	.000772	.001930
.000473	.001201	.000573	.001445	.000673	.001689	.000773	.001934
.000474	.001205	.000574	.001449	.000674	.001693	.000774	.001937
.000475	.001209	.000575	.001453	.000675	.001697	.000775	.001941
.000476	.001213	.000576	.001457	.000676	.001701	.000776	.001945
.000477	.001216	.000577	.001461	.000677	.001705	.000777	.001949

APPENDIX C

DATA 620/i ASSEMBLY SYSTEM

C-1. DATA 620/i INSTRUCTIONS IN NUMERIC ORDER

The following tables list the DATA 620/i instructions in numeric order. The octal codes, the corresponding DAS mnemonic (if any) a

brief definition and references to more detailed descriptions are included in this table. Not all 32,768 possible codes are shown here, the portions of the code that refer to addresses, counts (shifts), and external I/O devices are not included. The I/O device and function codes are shown separately in Appendix D.

C-1 DATA 620/i Instructions (Numeric Order)

Code	Mnemonic	Definition	References (Page)
000XXX	HLT	Halt, suspend instruction execution	31
001XXX	JMP, JIF	Jump	40-42
002XXX	JMPM, JIFM	Jump and Mark	43-44
003XXX	XEC, XIF	Execute	45-47
004XXX		Shift	31-35
005XXX		Register Interchange	35-38
00600X			
00601X	*LDAE, LDAI	Load into A Register	47,51
006020	*LDBE, LDBI	Load into B Register	48,52
00603X	*LDXE, LDXI	Load into X Register	48,52
00604X	*INRE, INRI	Increment Memory	49,54
00605X	*STAE, STAI	Store A Register	48,52

*Optional Instructions

C-1 DATA 620/i Instructions (Numeric Order)

Code	Mnemonic	Definition	References (Page)
00606X	*STBE, STBI	Store B Register	48,52
00607X	*STXE, STXI	Store X Register	48,52
00610X			
00611X	*ORAE, ORAI	Inclusive-OR to A Register	50,54
00612X	*ADDE, ADDI	Add to A Register	49,52
00613X	*ERAE, ERAI	Exclusive-OR to A Register	51,54
00614X	*SUBE, SUBI	Subtract from A Register	49,53
00615X	*ANAE, ANAI	AND to A Register	51,55
00616X	*MULE, MULI	Multiply M·B+A	49,53
00617X	*DIVE, DIVI	Divide (A,B)/M	50,53
00740X	SOF, ROF	Set Overflow to 1 or 0	31
007XXX		Undefined	
01XXXX	LDA	Load into A Register	25
02XXXX	LDB	Load into B Register	25
03XXXX	LDX	Load into X Register	25
04XXXX	INR	Increment Memory	27
05XXXX	STA	Store A Register	25
06XXXX	STB	Store B Register	25
07XXXX	STX	Store X Register	27

*Optional Instructions

C-1 DATA 620/i Instructions (Numeric Order)

Code	Mnemonic	Definition	References (Page)
100XXX	EXC, SEL	External Control or Select	59
101XXX	SEN	External Sense	60
1020XX	IME	Input to Memory	61
1021XX	INA	Input Inclusive-OR to A Register	61
1022XX	INB	Input Inclusive-OR to B Register	61
1023XX	INAB	Input Inclusive-OR to A and B Registers	
1024XX			
1025XX	CIA	Input into A Register	60
1026XX	CIB	Input into B Register	61
1027XX	CIAB	Input into both A and B Registers	
1030XX	OME	Output from Memory	62
1031XX	OAR	Output from A Register	61
1032XX	OBR	Output from B Register	62
1033XX	OAB	Output Inclusive-OR of A and B Registers	
1034XX		Undefined	
1035XX		Undefined	
1036XX		Undefined	
1037XX		Undefined	
104XXX		Auxiliary External Control	

C-1 DATA 620/i Instructions (Numeric Order)

Code	Mnemonic	Definition	References (Page)
105XXX		Undefined	
106XXX		Undefined	
107XXX		Undefined	
11XXX	ORA	Inclusive-OR to A Register	29
12XXX	ADD	Add to A Register	27
13XXX	ERA	Exclusive-OR to A Register	30
14XXX	SUB	Subtract from A Register	27
15XXX	ANA	AND to A Register	30
16XXX	*MUL	Multiply M·B+A	27
17XXX	*DIV	Divide (A,B)/M	29

*Optional Instructions

C-2 DATA 620/i Instructions (Alphabetical Order)

Mnemonic	Octal	Description	WDS/ Inst	Time Cycles	Indirect Address	References (Page)
ADD	120000	Add to A Register	1	2	Yes	27
*ADDE	006120	Add to A Register Extended	2	3	Yes	49
ADDI	006120	Add to A Register Immediate	2	2	No	52
ANA	150000	AND to A Register	1	2	Yes	30

*Optional Instructions

C-2 DATA 620/i Instructions (Alphabetical Order)

Mnemonic	Octal	Description	WDS/ Inst	Time Cycles	Indirect Address	References (Page)
*ANAE	006150	AND to A Register Extended	2	3	Yes	51
ANAI	006150	AND to A Register Immediate	2	2	No	55
AØFA	005511	Add OF to A Register	1	1	No	38
AØFB	005522	Add OF to B Register	1	1	No	38
AØFX	005544	Add OF to X Register	1	1	No	38
ASLA	004200+n	Arithmetic Shift Left A n Places	1	1+0.25n	No	34
ASLB	004000+n	Arithmetic Shift Left B n Places	1	1+0.25n	No	34
ASRA	004300+n	Arithmetic Shift Right A n Places	1	1+0.25n	No	33
ASRB	004100+n	Arithmetic Shift Right B n Places	1	1+0.25n	No	34
CIA	102500	Clear and Input to A Register	1	2	No	60
CIAB	102700	Clear and Input to A and B	1	2	No	
CIB	102600	Clear and Input to B Register	1	2	No	61
CPA	005211	Complement A Register	1	1	No	36
CPB	005222	Complement B Register	1	1	No	36
CPX	005244	Complement X Register	1	1	No	36

*Optional Instructions

C-2 DATA 620/i Instructions (Alphabetical Order)

Mnemonic	Octal	Description	WDS/ Inst	Time Cycles	Indirect Address	References (Page)
DAR	005311	Decrement A Register	1	1	No	36
DBR	005322	Decrement B Register	1	1	No	36
*DIV	170000	Divide AB Register				
		16-Bit	1	10-14	Yes	29
		18-Bit	1	11-15		
*DIVE	006170	Divide AB Register Extended				
		16-Bit	2	11-15	Yes	50
		18-Bit		12-16		
*DIVI	006170	Divide AB Register Immediate				
		16-Bit	2	10-14	No	53
		18-Bit		11-15		
DXR	005344	Decrement X Register	1	1	No	36
ERA	130000	Exclusive OR to A Register	1	2	Yes	30
*ERAE	006130	Exclusive OR to A Register Extended	2	3	Yes	51
ERAI	006130	Exclusive OR to A Register Immediate	2	2	No	54
EXC	100000	External Control Function	1	1	No	59
HLT	000000	Halt	1	1	No	31
IAR	005111	Increment A Register	1	1	No	36
IBR	005122	Increment B Register	1	1	No	36
IME	102000	Input to Memory	2	3	No	36
INA	102100	Input to A Register	1	2	No	61
INAB	102300	Input to A and B Registers	1	2	No	61

*Optional Instructions

C-2 DATA 620/i Instructions (Alphabetical Order)

Mnemonic	Octal	Description	WDS/ Inst	Time Cycles	Indirect Address	References (Page)
INB	102200	Input to B Register	1	2	No	61
INR	040000	Increment and Replace	1	3	Yes	27
*INRE	006040	Increment and Replace Extended	2	4	Yes	49
INRI	006040	Increment and Replace Immediate	2	3	No	54
IXR	005144	Increment X Register	1	1	No	36
JAN	001004	Jump if A Register Negative	2	2	Yes	40
JANM	002004	Jump and Mark if A Register Negative	2	2-3	Yes	43
JAP	001002	Jump if A Register Positive	2	2	Yes	40
JAPM	002002	Jump and Mark if A Register Positive	2	2-3	Yes	43
JAZ	001010	Jump if A Register Zero	2	2	Yes	42
JAZM	002010	Jump and Mark if A Register	2	2-3	Yes	44
JBZ	001020	Jump if B Register Zero	2	2	Yes	42
JBZM	002020	Jump and Mark if B Register Zero	2	2-3	Yes	44
JMP	001000	Jump Unconditionally	2	2	Yes	40

*Optional Instructions

C-2 DATA 620/i Instructions (Alphabetical Order)

Mnemonic	Octal	Description	WDS/ Inst	Time Cycles	Indirect Address	References (Page)
JMPM	002000	Jump and Mark if Unconditionally	2	3	Yes	43
JØF	001001	Jump if Overflow On	2	2	Yes	40
JØFM	002001	Jump and Mark if Overflow On	2	2-3	Yes	43
JS1M	002100	Jump and Mark if Sense Switch 1 On	2	2-3	Yes	44
JS2M	002200	Jump and Mark if Sense Switch 2 On	2	2-3	Yes	44
JS3M	002400	Jump and Mark if Sense Switch 3 On	2	2-3	Yes	44
JSS1	001100	Jump if Sense Switch 1 On	2	2	Yes	42
JSS2	001200	Jump if Sense Switch 2 On	2	2	Yes	42
JSS3	001400	Jump if Sense Switch 3 On	2	2	Yes	42
JXZ	001040	Jump X Register Zero	2	2	Yes	42
JXZM	002040	Jump and Mark X Zero	2	2-3	Yes	44
LASL	004400+ <i>n</i>	Long Arithmetic Shift Left <i>n</i> Places	1	1+0.50 <i>n</i>	No	34
LASR	004500+ <i>n</i>	Long Arithmetic Shift Right <i>n</i> Places	1	1+0.50 <i>n</i>	No	34
LDA	010000	Load A Register	1	2	Yes	25

C-2 DATA 620/i Instructions (Alphabetical Order)

Mnemonic	Octal	Description	WDS/ Inst	Time Cycles	Indirect Address	References (Page)
*LDAE	006010	Load A Register Extended	2	3	Yes	47
LDAI	006010	Load A Register Immediate	2	2	No	51
LDB	020000	Load B Register	1	2	Yes	25
*LDBE	006020	Load B Register Extended	2	3	Yes	48
LDBI	006020	Load B Register Immediate	2	2	No	52
LDX	030000	Load X Register	1	2	Yes	25
*LDXE	006030	Load X Register Extended	2	3	Yes	48
LDXI	006030	Load X Register Immediate	2	2	No	52
LLRL	004440+n	Long Logical Rotate Left n Places	1	1+0,50n	No	33
LLSR	004540+n	Long Logical Shift Right n Places	1	1+0,50n	No	33
LRLA	004240+n	Logical Rotate Left A n Places	1	1+0,25n	No	33
LRLB	004040+n	Logical Rotate Left B n Places	1	1+0,25n	No	33
LSRA	004340+n	Logical Shift Right A n Places	1	1+0,25n	No	31

*Optional Instructions

C-2 DATA 620/i Instructions (Alphabetical Order)

Mnemonic	Octal	Description	WDS/ Inst	Time Cycles	Indirect Address	References (Page)
LSRB	004140 ^{tn}	Logical Shift Right B n Places	1	1+0.25n	No	33
*MUL	160000	Multiply B Register 16-Bit 18-Bit	1	10 11	Yes	27
*MULE	006160	Multiply B Register Extended 16-Bit 18-Bit	2	11 12	Yes	49
*MULI	006160	Multiply B Register Immediate 16-Bit 18-Bit	2	10 11	No	53
NØP	00500	No Operation	1	1	No	31
ØAB	103300	Output OR of A and B Registers	1	2	No	
ØAR	103100	Output from A Register	1	2	No	61
ØBR	103200	Output from B Register	1	2	No	62
ØME	103000	Output from Memory	2	3	No	62
ØRA	110000	Inclusive OR to A Register	1	2	Yes	29
*ØRAE	006110	Inclusive OR to A Register Extended	2	3	Yes	50
ØRAI	006110	Inclusive OR to A Register Immediate	2	2	No	54
RØF	007400	Reset Overflow	1	1	No	31

*Optional Instructions

C-2 DATA 620/i Instructions (Alphabetical Order)

Mnemonic	Octal	Description	WDS/ Inst	Time Cycles	Indirect Address	References (Page)
SEN	101000	Sense Input/Output Lines	2	2.25	Yes	60
SØF	007401	Set Overflow	1	1	No	31
SØFA	005711	Subtract OFLO from A Register	1	1	No	38
SØFB	005722	Subtract OFLO from B Register	1	1	No	38
SØFX	005744	Subtract OFLO from X Register	1	1	No	38
STA	050000	Store A Register	1	2	Yes	25
*STAE	006050	Store A Register Extended	2	3	Yes	48
STAI	006050	Store A Register Immediate	2	2	No	52
STB	060000	Store B Register	1	2	Yes	25
*STBE	006060	Store B Register Extended	2	3	Yes	48
STBI	006060	Store B Register Immediate	2	2	No	52
STX	070000	Store X Register	1	2	Yes	27
*STXE	006070	Store X Register Extended	2	3	Yes	48
STXI	006070	Store X Register Immediate	2	2	No	52

*Optional Instructions

C-2 DATA 620/i Instructions (Alphabetical Order)

Mnemonic	Octal	Description	WDS/ Inst	Time Cycles	Indirect Address	References (Page)
SUB	140000	Subtract from A Register	1	2	Yes	27
*SUBE	006140	Subtract from A Register Extended	2	3	Yes	49
SUBI	006140	Subtract from A Register Immediate	2	2	No	53
TAB	005012	Transfer A to B Register	1	1	No	37
TAX	005014	Transfer A to X Register	1	1	No	37
TBA	005021	Transfer B to A Register	1	1	No	37
TBX	005024	Transfer B to X Register	1	1	No	37
TXA	005041	Transfer X to A Register	1	1	No	37
TXB	005042	Transfer X to B Register	1	1	No	37
TZA	005001	Transfer Zero to A Register	1	1	No	37
TZB	005002	Transfer Zero to B Register	1	1	No	38
TZX	005004	Transfer Zero to X Register	1	1	No	38
XAN	003004	Execute A Register Negative	2	2	Yes	46

*Optional Instructions

C-2 DATA 620/i Instructions (Alphabetical Order)

Mnemonic	Octal	Description	WDS/ Inst	Time Cycles	Indirect Address	References (Page)
XAP	003002	Execute A Register Positive	2	2	Yes	45
XAZ	003010	Execute A Register Zero	2	2	Yes	46
XBZ	003020	Execute B Register Zero	2	2	Yes	46
XEC	003000	Execute Unconditionally	2	2	Yes	45
XØF	003001	Execute Overflow Set	2	2	Yes	45
XS1	003100	Execute SENSE Switch 1 Set	2	2	Yes	46
XS2	003200	Execute SENSE Switch 2 Set	2	2	Yes	47
XS3	003400	Execute SENSE Switch 3 Set	2	2	Yes	47
XXZ	003040	Execute X Register Zero	2	2	Yes	46

C-3 Single-Word Addressed Instructions

C-3 (a) Load/Store Instruction Group

Op Code		Instruction	Timing (Cycles)	References (Page)
Octal	Mnemonic			
01	LDA	Load A Register	2	25
02	LDB	Load B Register	2	25
03	LDX	Load X Register	2	25
05	STA	Store A Register	2	25
06	STB	Store B Register	2	25
07	STX	Store X Register	2	27

C-3 Single-Word Addressed Instructions

C-3 (b) Arithmetic Instruction Group

Op Code		Instruction	Timing (Cycles)	References (Page)
Octal	Mnemonic			
04	INR	Increment and Replace	3	27
12	ADD	Add Memory to A	2	27
14	SUB	Subtract Memory from A	2	27
16	*MUL	Multiply 16-bit	10	27
		18-bit	11	
17	*DIV	Divide 16-bit	10-14	29
		18-bit	11-15	

*Optional Instructions

C-3 (c) Logical Instruction Group

Op Code		Instruction	Timing (Cycles)	References (Page)
Octal	Mnemonic			
11	ØRA	Inclusive OR, Memory and A	2	29
13	ERA	Exclusive OR, Memory and A	2	30
15	ANA	AND Memory and A	2	30

C-3 (d) Addressing Modes for Single Word Addressed Instructions

m Field			Addressing Mode	Operation
11	10	9		
0	X	X	Direct	Combine bits 9, 10 with a field (0-8) to form effective address (0000 - 2407).
1	0	0	Relative	Add a field (bits 0-8) to contents of P to form effective address (Mod 2^{15}).
1	0	1	Index (X Register)	Add a field (bits 0-8) to contents of X to form effective address (Mod 2^{15}).

C-3 Single-Word Addressed Instructions

m Field			Addressing Mode	Operation
11	10	9		
1	1	0	Index (B Register)	Add a field (bits 0-8) to contents of B to form effective address (Mod 2^{15}).
1	1	1	Indirect	a field (bits 0-8) specifies location of an address word.

C-4 Control Instruction Group Codes (Single-Word, Non-Addressable)

Op Code		m Field	M Field	Instruction	Timing (Cycles)	References (Page)
Octal	Mnemonic					
00	HLT	0	XXX	Halt	1	31
00	NØP	5	000	No Operation	1	31
00	RØF	7	400	Reset Overflow	1	31
00	SØF	7	401	Set Overflow	1	31

C-5 Shift Instruction Group

C-5(a) Instruction Format

Octal	Octal	M Field								
OP Code	m Field	U ₈	U ₇	U ₆	U ₅	U ₄	U ₃	U ₂	U ₁	U ₀
00	4	0 = A or B 1 = A & B	0 = B 1 = A	0 = Left 1 = Right	0 = Arith. 1 = Logical rotate	Shift Count (0 - 31)				

C-5 (b) Instruction Format

U ₈	U ₇	U ₆	U ₅	Mnemonic	Shift Instruction	Timing (Cycles)	References (Page)
0	0	0	0	ASLB	Arithmetic Shift B Left	1 + 0.25n	34
0	0	0	1	LRLB	Logical Rotate B Left	1 + 0.25n	33
0	0	1	0	ASRB	Arithmetic Shift B Right	1 + 0.25n	34
0	0	1	1	LSRB	Logical Shift B Right	1 + 0.25n	33
0	1	0	0	ASLA	Arithmetic Shift A Left	1 + 0.25n	34
0	1	0	1	LRLA	Logical Rotate A Left	1 + 0.25n	33
0	1	1	0	ASRA	Arithmetic Shift A Right	1 + 0.25n	33
0	1	1	1	LSRA	Logical Shift A Right	1 + 0.25n	31
1	0	0	0	LASL	Long Arithmetic Shift A, B Left	1 + 0.50n	34
1	0	0	1	LLRL	Long Logical Rotate A, B Registers Left	1 + 0.50n	33
1	0	1	0	LASR	Long Arithmetic Shift A, B Right	1 + 0.50n	34
1	0	1	1	LLSR	Long Logical Shift, A, B Registers	1 + 0.50n	33
1	1	0	0		Invalid		
1	1	0	1		Invalid		
1	1	1	0		Invalid		
1	1	1	1		Invalid		

C-6 Register Change Instruction Group

C-6 (a) Instruction Format

Octal		M Field									
		Source						Dest.			
Class Code	m Field	U ₈	U ₇	U ₆	U ₅	U ₄	U ₃	U ₂	U ₁	U ₀	Type of Transfer
00	5	0F	0 0 1 1	0 1 0 1	X	B	A	X	B	A	Transfer Unchanged Transfer Incremented Transfer Complemented Transfer Decrement

Note: Multiple source transfer results in inclusive-OR; multiple source complemented results in complement inclusive-OR.

0F is the conditional indicator. If 0F is zero, the instruction is executed unconditionally. If 0F is one, the instruction is executed only if the overflow is on.

C-6 (b) Register Change Instruction Codes

Class Code Field Octal	Mnemonic	Register Change Instruction	Timing	References (Page)
0 0 1	TZA	Transfer Zero to A Register	1	37
0 0 2	TZB	Transfer Zero to B Register	1	37
0 0 4	TZX	Transfer Zero to X Register	1	38
0 1 2	TAB	Transfer A Register to B Register	1	37
0 1 4	TAX	Transfer A Register to X Register	1	37
0 2 1	TBA	Transfer B Register to A Register	1	37
0 2 4	TBX	Transfer B Register to X Register	1	37
0 4 1	TXA	Transfer X Register to A Register	1	37
0 4 2	TXB	Transfer X Register to B Register	1	37
1 1 1	IAR	Increment A Register	1	36
1 2 2	IBR	Increment B Register	1	36
1 4 4	IXR	Increment X Register	1	36
3 1 1	DAR	Decrement A Register	1	36
3 2 2	DBR	Decrement B Register	1	36
3 4 4	DXR	Decrement X Register	1	36
5 1 1	AØFA	Add Overflow to A Register	1	38
5 2 2	AØFB	Add Overflow to B Register	1	38
5 4 4	AØFX	Add Overflow to X Register	1	38
7 1 1	SØFA	Subtract Overflow from A Register	1	38
7 2 2	SØFB	Subtract Overflow from B Register	1	38
7 4 4	SØFX	Subtract Overflow from X Register	1	38

C-7 Sequence Change Instruction Group

C-7 (a) Instruction Format

Octal		M Field									References (Page)
OP Code	m Field	U ₈	U ₇	U ₆	U ₅	U ₄	U ₃	U ₂	U ₁	U ₀	
00	1, 2, or 3	SS3 ON	SS2 ON	SS1 ON	X = 0	B = 0	A = 0	A < 0	A ≥ 0	OF = 1	39

Note: Test condition is logical AND of all a field bits.

The action taken is Jump if m = 1, Jump and Mark if M = 2, Execute if m = 3.

C-7 (b) Conditional Instruction Codes

M Field Octal	Mnemonic			Condition	Timing (Cycles)	References (Page)
	m = 1	m = 2	m = 3			
	JIF	JMIF	XIF	General	*2	
0 0 0	JMP	JMPM	XEC	Unconditionally	2	40,43,45
0 0 1	JØF	JØFM	XØF	If Overflow Set	2	40,43,45
0 0 2	JAP	JAPM	XAP	If A Register Positive	2	40,43,45
0 0 4	JAN	JANM	XAN	If A Register Negative	2	40,43,46
0 1 0	JAZ	JAZM	XAZ	If A Register Zero	2	42,44,46
0 2 0	JBZ	JBZM	XBZ	If B Register Zero	2	42,44,46
0 4 0	JXZ	JXZM	XXZ	If X Register Zero	2	42,44,46
1 0 0	JSS1	JS1M	XS1	If SENSE Switch 1 Set	2	42,44,46
2 0 0	JSS2	JS2M	XS2	If SENSE Switch 2 Set	2	42,44,47
4 0 0	JSS3	JS3M	XS3	If SENSE Switch 3 Set	2	42,44,47

*For Jump and Mark instructions, an additional memory cycle is used if the test condition is true.

C-8 Extended Address Instruction Group (Optional)

Basic OP Code		Effective OP Code		Instruction	Timing (Cycles)	References (Page)
Octal	Mnemonic	m Field	M Field			
00	LDAE	6	01X	Load A Register Extended	3	47
00	LDBE	6	02X	Load B Register Extended	3	48
00	LDXE	6	03X	Load X Register Extended	3	48
00	INRE	6	04X	Increment and Replace Extended	4	49
00	STAE	6	05X	Store A Register Extended	3	48
00	STBE	6	06X	Store B Register Extended	3	48
00	STXE	6	07X	Store X Register Extended	3	48
00	ØRAE	6	11X	Inclusive OR Extended	3	50
00	ADDE	6	12X	Add Memory to A Register Extended	3	49
00	ERAЕ	6	13X	Exclusive OR Extended	3	51
00	SUBE	6	14X	Subtract Memory from A Register Extended	3	49
00	ANAE	6	15X	AND Extended	3	51

C-8 Extended Address Instruction Group (Optional)

Basic OP Code		Effective OP Code		Instruction	Timing (Cycles)	References (Page)
Octal	Mnemonic	m Field	M Field			
00	MULE	6	16X	Multiply 16-Bit Extended Multiply 18-Bit Extended	11 12	49
00	DIVE	6	17X	Divide 16-Bit Extended Divide 18-Bit Extended	11 - 15 12 - 16	50

X Addressing

4 Relative to P

5 Relative to X

6 Relative to B

7 Direct or Indirect (depending on bit 15 of the address word)

C-9 Immediate Instruction Group

Basic OP Code		Effective OP Code (octal)		Instruction	Timing (Cycles)	References (Page)
Octal	Mnemonic	m Field	M Field			
00	LDAI	6	010	Load A Immediate	2	51
00	LDBI	6	020	Load B Immediate	2	52
00	LDXI	6	030	Load X Immediate	2	52
00	INRI	6	040	Increment and Replace Immediate	2	54
00	STAI	6	050	Store A Immediate	2	52
00	STBI	6	060	Store B Immediate	2	52
00	STXI	6	070	Store X Immediate	2	52
00	ORAI	6	110	Inclusive OR Immediate	2	54
00	ADDI	6	120	Add Immediate	2	52
00	ERAI	6	130	Exclusive OR Immediate	2	54
00	SUBI	6	140	Subtract Immediate	2	53
00	ANAI	6	150	AND Immediate	2	55/56
00	*MULI	6	160	Multiply Immediate 16 bits	10	53
				18 bits	11	
00	*DIVI	6	170	Divide Immediate 16 bits	10 - 14	53
				18 bits	11 - 15	

*Optional Instructions

C-10 Input/Output Instruction Group

OP Code		Octal		Instruction	Timing (Cycles)	References (Page)
Octal	Mnemonic	M Field	A Field			
10	EXC	0	XZZ	External Control	1	59
10	SEN	1	XZZ	Program Sense	2	60
10	IME	2	0ZZ	Input to Memory	3	61
10	INA	2	1ZZ	Input to A	2	61
10	INB	2	2ZZ	Input to B	2	61
10	INAB	2	3ZZ	Input to A and B	2	
10	CIA	2	5ZZ	Clear and Input to A	2	60
10	CIB	2	6ZZ	Clear and Input to B	2	61
10	CIAB	2	7ZZ	Clear and Input to A and B	2	
10	ØME	3	0ZZ	Output from Memory	3	62
10	ØAR	3	1ZZ	Output from A	2	61
10	ØBR	3	2ZZ	Output from B	2	62
10	ØAB	3	3ZZ	Output inclusive-OR or A and B	2	

X - Mode or logical unit number

ZZ - Device number (See Appendix D)

APPENDIX D

I/O CONTROL CODES AND DEVICE ADDRESSES

Table D-1. Interrupt Module Instruction Codes

The following instruction codes are for use with the first interrupt module. Device addresses 40₈ through 47₈ are reserved for interrupt modules.

Mnemonic	Octal	Function
A. External Control		
*EXC140	100140	Clear AC Register
EXC 240	100240	Enable Interrupt Module
EXC 440	100440	Inhibit Interrupt Module
EXC 540	100540	Initialize Interrupt Module
B. Transfer		
OME 40	103040	Load Mask from Memory
OAR 40	103140	Load Mask from A Register
OBR 40	103240	Load Mask from B Register
C. Sense		
None		
*Available only with option accepting Alternating Current interrupts. This instruction clears the register that distinguishes the AC interrupt level.		

Table D-2. Buffer Interface Controller Instruction Codes

The following instruction codes are for use with the first buffer interface controller. Device addresses 20_h through 27_h are reserved for this use.

Mnemonic	Octal	Function
A. External Control		
EXC 020	100020	Activate Enable
EXC 021	100021	Initialize
B. Transfer		
ØAR 20	103120	Load Initial Register from A
ØBR 20	103220	Load Initial Register from B
ØME 20	103020	Load Initial Register from Memory
ØAR 21	103121	Load Final Register from A
ØBR 21	103221	Load Final Register from B
ØME 21	103021	Load Final Register from Memory
INA 20	102120	Read Initial Register into A
INB 20	102220	Read Initial Register into B
IME 20	102020	Read Initial Register into Memory
CIA 20	102520	Read Initial Register into Cleared A
CIB 20	102620	Read Initial Register into Cleared B
C. Sense		
SEN 20	101020	Sense BIC Not Busy
SEN 21	101021	Sense Abnormal Device Stop

Table D-3. Teletype Instruction Codes
D-3 (a) Model A Teletype Instructions

Mnemonic	Octal	Function
A. External Control		
EXC 000	100000	Select High-Speed Input
EXC 100	100100	Select Paper Tape Input
EXC 200	100200	Select Keyboard Input
EXC 300	100300	Select Page and/or Paper Tape Out
EXC 400	100400	Select Off
B. Transfer		
OAR 00	103100	Transfer A Register to TTY Buffer
OBR 00	103200	Transfer B Register to TTY Buffer
OME 00	103000	Transfer Memory to TTY Buffer
INA 00	102100	Transfer TTY Buffer to A Register
INB 00	102200	Transfer TTY Buffer to B Register
IME 00	102000	Transfer TTY Buffer to Memory
CIA 00	102500	Transfer TTY Buffer to A Register cleared
CIB 00	102600	Transfer TTY Buffer to B Register cleared
C. Sense		
SEN 000	101000	Sense TTY Not Busy
SEN 100	101100	Sense TTY Buffer Ready
SEN 300	101300	Sense TTY Reader Ready

The following are A-type teletypes:

620-60A

Table D-3. Teletype Instruction Codes (Continued)
D-3 (b) Model B Teletype Instructions

Mnemonic	Octal	Function	
A. External Control			
EXC 101	100101	Connect Write Register to BIC	
EXC 201	100201	Connect Read Register to BIC	
EXC 401	100401	Initialize	
B. Transfer			
OAR 01	103101	Transfer A Register to Write Register	
OBR 01	103201	Transfer B Register to Write Register	
OME 01	103001	Transfer Memory Register to Write Register	
IAR 01	102101	Transfer Read Register to A Register	
IBR 01	102201	Transfer Read Register to B Register	
IME 01	102001	Transfer Read Register to Memory Register	
CIA 01	102501	Transfer Read Register to Cleared A Register	
CIB 01	102601	Transfer Read Register to Cleared B Register	
C. Sense			
SEN 101	101101	Write Register Ready	
SEN 201	101201	Read Register Ready	
D. Teletype Command Codes			
Function	Symbol	Code	Typed As
Print Enable	SOM	201	Control and A
Print Suppress	EOT	204	Control and D
Reader On	XON	221	Control and Q
Punch On	TAPE	222	Control and R
Reader Off	XOFF	223	Control and S
Punch Off	TAPE OFF	224	Control and T

The following models are B-type teletypes:

620-60B (ASR-33 TM)
620-61B (ASR-35 TM)
620-62B (KSR-35 TM)

Note: External control instructions are for use only with the Buffer Interlace Controller.

Table D-4. Card Reader Instruction Codes

The following instruction codes are for use with the 90 CPM or 1100 CPM card reader. For additional card readers, device addresses will be assigned at the time of system definition.

Mnemonic	Octal	Function
A. External Control		
EXC 230	100230	Read One Card
*EXC 630	100630	Step Read One Character
B. Transfer		
INA 30	102130	Transfer to A Register
INB 30	102230	Transfer to B Register
IME 30	102030	Transfer to Memory
CIA 30	102530	Transfer to A Register Cleared
CIB 30	102630	Transfer to B Register Cleared
C. Sense		
SEN 130	101130	Sense Character Ready
*SEN 230	101230	Sense Reader Not Busy
SEN 630	101630	Sense Reader Ready
*Delete for 1100 CPM reader.		

Table D-5. Gated-Input-Channel Instruction Codes

The following instruction codes are for use with the gated input channel. Device addresses for additional input channels will be assigned at the time of system definition.

Mnemonic	Octal	Function
A. External Control		
None		
B. Transfer		
INA 60	102160	Input from Channel to A Register
INB 60	102260	Input from Channel to B Register
IME 60	102060	Input from Channel to Memory
CIA 60	102560	Input from Channel to Cleared A Register
CIB 60	102660	Input from Channel to Cleared B Register
C. Sense		
SEN 460	101460	Sense Transfer in Request

Table D-6. Buffer-Input-Channel Instruction Codes

The following instruction codes are for use with the buffer input channel. Device addresses for additional input channels will be assigned at the time of system definition.

Mnemonic	Octal	Function
A. External Control		
None		
B. Transfer		
INA 62	102162	Input from Channel to A Register
INB 62	102262	Input from Channel to B Register
IME 62	102062	Input from Channel to Memory
CIA 62	102562	Input from Channel to Cleared A Register
CIB 62	102662	Input from Channel to Cleared B Register
C. Sense		
SEN 462	101462	Sense Transfer in Request

Table D-7. Gated-Output-Channel Instruction Codes

The following instruction codes are for use with the gated output channel. Device addresses for additional output channels will be assigned at the time of system definition.

Mnemonic	Octal	Function
A. External Control		
None		
B. Transfer		
ØAR 60	103160	Output from A Register through Channel
ØBR 60	103260	Output from B Register through Channel
ØME 60	103060	Output from Memory through Channel
C. Sense		
SEN 260	101260	Sense Data Request

Table D-8. Buffer-Output-Channel Instruction Codes

The following codes are for use with the buffer output channel. Device addresses for additional output channels will be assigned at the time of system definition.

Mnemonic	Octal	Function
A. External Control		
None		
B. Transfer		
ØAR 62	103162	Output from A Register through Channel
ØBR 62	103262	Output from B Register through Channel
OME 62	103062	Output from Memory through Channel
C. Sense		
SEN 262	101262	Sense Data Request

Table D-9. High-Speed Paper Tape I/O Instruction Codes

The following instruction codes are for use with the paper tape I/O unit. For additional units, device addresses will be assigned at the time of system definition. If only a reader or a punch is attached, use only those codes which apply.

Mnemonic	Octal	Function
A. External Control		
EXC 037	100037	Connect Punch to BIC
EXC 437	100437	Stop Reader
EXC 537	100537	Start Reader
EXC 637	100637	Punch Buffer
EXC 737	100737	Read One Character
B. Transfer		
OAR 37	103137	Load Buffer from A Register
OBR 37	103237	Load Buffer from B Register
OME 37	103037	Load Buffer from Memory
INA 37	102137	Read Buffer into A Register
INB 37	102237	Read Buffer into B Register
IME 37	102037	Read Buffer into Memory
CIA 37	102537	Read Buffer into Cleared A Register
CIB 37	102637	Read Buffer into Cleared B Register
C. Sense		
SEN 537	101537	Sense Buffer Ready

Table D-10. Magnetic Tape Unit Instruction Codes

The following instruction codes are for use with the first magnetic tape unit. Device addresses 10g through 13g are reserved for other magnetic tape.

Mnemonic	Octal	Function
A. External Control		
EXC 010	100010	Read One Record Binary
EXC 110	100110	Read One Record BCD
EXC 210	100210	Write One Record Binary
EXC 310	100310	Write One Record BCD
EXC 410	100410	Write File Mark
EXC 510	100510	Forward One Record
EXC 610	100610	Backspace One Record
EXC 710	100710	Rewind
B. Transfer		
ØAR 10	103110	Load Buffer from A Register
ØBR 10	103210	Load Buffer from B Register
ØME 10	103010	Load Buffer from Memory
INA 10	102110	Read Buffer into A Register
INB 10	102210	Read Buffer into B Register
IME 10	102010	Read Buffer into Memory
CIA 10	102510	Read Buffer into Cleared A Register
CIB 10	102610	Read Buffer into Cleared B Register
C. Sense		
SEN 010	101010	Sense Parity Error
SEN 110	101110	Sense Buffer Ready
SEN 210	101210	Sense MTU Ready
SEN 310	101310	Sense File Mark
SEN 410	101410	Sense High Density
SEN 510	101510	Sense End of Tape
SEN 610	101610	Sense Beginning of Tape
SEN 710	101710	Sense Rewinding

APPENDIX E

STANDARD CHARACTER CODES

Table E-1. DATA 620/i Standard Codes

Symbol	ASCII	Printer	Mag Tape	Hollerith	FORTRAN
@	300	00	32	0-2-8	77
A	301	01	61	12-1	13
B	302	02	62	12-2	14
C	303	03	63	12-3	15
D	304	04	64	12-4	16
E	305	05	65	12-5	17
F	306	06	66	12-6	20
G	307	07	67	12-7	21
H	310	10	70	12-8	22
I	311	11	71	12-9	23
J	312	12	41	11-1	24
K	313	13	42	11-2	25
L	314	14	43	11-3	26
M	315	15	44	11-4	27
N	316	16	45	11-5	30
O	317	17	46	11-6	31
P	320	20	47	11-7	32
Q	321	21	50	11-8	33
R	322	22	51	11-9	34
S	323	23	22	0-2	35
T	324	24	23	0-3	36

Table E-1. DATA 620/i Standard Codes (Continued)

Symbol	ASCII	Printer	Mag Tape	Hollerith	FORTRAN
U	325	25	24	0-4	37
V	326	26	25	0-5	40
W	327	27	26	0-6	41
X	330	30	27	0-7	42
Y	331	31	30	0-8	43
Z	332	32	31	0-9	44
[	333	33	75	12-5-8	76*
\	334	34	36	0-6-8	76*
]	335	35	55	11-5-8	76*
↑	336	36	17	7-8	76*
			(Note)		
←	337	37	20	2-8	76 ¹
blank	240	40	20	No Punch	00
!	241	41	52	11-2-8	51
"	242	42	35	0-5-8	62
#	243	43	37	0-7-8	63
\$	244	44	53	11-3-8	60
%	245	45	57	11-7-8	64
&	246	46	77	12-7-8	65
'	247	47	14	4-8	66
(	250	50	34	0-4-8	52
)	251	51	74	12-4-8	53

Note: End-of-file for mag tape.

*: Undefined character.

1: Form control: Return to col 1. FORTRAN System only.

Table E-1. DATA 620/i Standard Codes (Continued)

Symbol	ASCII	Printer	Mag Tape	Hollerith	FORTRAN
*	252	52	54	11-4-8	47
+	253	53	60	12	45
,	254	54	33	0-3-8	54
-	255	55	40	11	46
.	256	56	73	12-3-8	51
/	257	57	21	0-1	50
0	260	60	12	0	01
1	261	61	01	1	02
2	262	62	02	2	03
3	263	63	03	3	04
4	264	64	04	4	05
5	265	65	05	5	06
6	266	66	06	6	07
7	267	67	07	7	10
8	270	70	10	8	11
9	271	71	11	9	12
:	272	72	15	5-8	67
;	273	73	56	11-6-8	70
<	274	74	76	12-6-8	76*
=	275	75	13	3-8	55
>	276	76	16	6-8	76 ²
?	277	77	72	12-2-8	76

*: Undefined character.

2: Tab control: Skip to col 7. FORTRAN System only.

Table E-2. Teletype Character Codes

Teletype Character	DATA 620/i Internal Code	Teletype Character	DATA 620/i Internal Code
0	260	Q	321
1	261	R	322
2	262	S	323
3	263	T	324
4	264	U	325
5	265	V	326
6	266	W	327
7	267	X	330
8	270	Y	331
9	271	Z	332
A	301	blank	240
B	302	!	241
C	303	"	242
D	304	#	243
E	305	\$	244
F	306	%	245
G	307	&	246
H	310	'	247
I	311	(	250
J	312	)	251
K	313	*	252
L	314	+	253
M	315	,	254
N	316	-	255
O	317	.	256
P	320	/	257

Table E-2. Teletype Character Codes (Continued)

Teletype Character	DATA 620/i Internal Code	Teletype Character	DATA 620/i Internal Code
:	272	V TAB	213
;	273	FORM	214
	274	RETURN	215
=	275	SO	216
	276	SI	217
?	277	DCO	220
@	300	X-ON	221
	333	TAPE AUX ON	222
	334	X-OFF	223
	335	TAPE AUX OFF	224
	336		
	337	ERROR	225
Delete	377	SYNC	226
NUL	200	LEM	227
SOM	201	S0	230
EOA	202	S1	231
EOM	203	S2	232
EOT	204	S3	233
WRU	205	S4	234
RU	206	S5	235
BEL	207	S6	236
FE	210	S7	237
H TAB	211		
LINE FEED	212		

APPENDIX F

DAS REFERENCE INFORMATION

Table F-1. DAS Pseudo Instruction Codes

Mnemonic Code	Meaning	Reference (Page)
BEGIN	Create a new assembler location counter	126
BES	Allocate a block of memory space (labeled at beginning)	127
BSS	Allocate a block of memory space (labeled at end)	127
CALL	Call for a subroutine with a parameter list	130
*COMN	Defines common blocks of memory space	134
CONT	Provide a target location that will not be saved	129
DATA	Defines constants in a program	126
DETL	List all source statements	131
DUP	Repeat the assembly of a few instructions	127
EJEC	Start a new page on the listing	131
END	Terminate the assembly process	129
ENTR	Provide a space for entry into a subroutine	130
EQU	Give a value to a label	123
*EXT	Specifies symbol is defined in another program	134
*FORT	Sets up compatibility with FORTRAN Loader	134
GOTO	Skip the assembly of several instructions	128
IFF	Conditional assembly test	128
IFT	Conditional assembly test	128
LIST	Ask for a program listing	130
*FORTRAN compatibility pseudo instructions		

Table F-1. DAS Pseudo Instruction Codes (Continued)

Mnemonic Code	Meaning	Reference (Page)
LOC	Give a starting value to a subroutine location	125
MAX	Give a label the value that is greatest of a set	124
MIN	Give a label the value that is least of a set	124
MORE	Interrupt assembler to insert new inputs	129
MZE	Define constants with the sign bit on	127
*NAME	Names the entry point(s) in a program	134
NLIS	Suppress program listing	130
NPUN	Suppress punching of the object program	131
OPSY	Define a new mnemonic instruction code	124
ORG	Give a starting location to an assembly	125
PUNC	Ask for punching of the object program	130
PZE	Define positive constants in program	127
READ	Define limits and codes for punch-card inputs	131
RETU	Provide for an exit from a subroutine	130
SET	Give a new value to a label	123
SMRY	Suppress listing unassembled source statements	131
SPAC	Skip lines on a page of listing	131
USE	Use location counter in variable field to assign locations	126
*FORTRAN compatibility pseudo instructions		

Table F-2. DAS Assembler Error Codes

Code	Meaning
*AD	Address expression in error.
*CH	Illegal character in source line.
*DC	A decimal character appears in an octal constant.
*DD	Illegal redefinition of a label or location counter.
*EX	Expression contains the illegal appearance of two consecutive arithmetic operators.
*FA	Floating-point format error.
*IL	The first non-blank character on a line is illegal, line not processed.
*MA	Inconsistent use of indexing and indirect addressing.
*NR	No room left in label table for this label.
*NS	Nested DUP statements.
*OP	The instruction code is undefined; a two-word gap is left in memory to allow patching.
*QQ	Illegal use of single quote.
*SE	Label value different from pass 1.
*SP	Illegal use of a special character for operand in address evaluation.
*SY	Expression contains an undefined label.
*SZ	Expression value too large for size of subfield.
*TF	Tag error, undefined or illegal index.
*UD	Undefined label in variable field of a USE instruction.
*VF	Instruction contains variable subfields either missing or inconsistent with the computer instruction type.
*XR	Address out of range for index specification.
*=	Illegal use of literal =.

APPENDIX G

STANDARD DATA 620/i SUBROUTINES

Subroutines	Locations	Time	References (Page)
Elementary Functions*			
Log ^e (1 + X), (0 ≤ X < 1)	19	365 μsec	176
Exponential (e ^{-X}) (0 ≤ X < 1)	17	283 μsec	180
Exponential (e ^{+X}) (0 ≤ X < 1)	17	333 μsec	178
Square Root (0 ≤ X < 1)	58	493 μsec	182
Sine X (-π < X < π)	31	315 μsec	184
Cosine X (-π < X < π)	20	310 μsec	186
Arctan (-1 to 1)	15	380 μsec	188
Single Precision (fixed point)			
Multiply (optional)	hardware	18 μsec	27
Multiply (programmed)	38	728 μsec	146
Divide (optional)	hardware	27 μsec	28
Divide (programmed)	70	360 μsec	147
Double Precision (fixed point)			
Addition	23	54.0 μsec	150
Subtraction	25	57.6 μsec	152
Multiply	38	97.2 μsec	154
Divide	55	310 μsec	156
Conversion			
Binary-to-BCD (4 characters)	31	249 μsec	142
BCD-to-Binary	28	205 μsec	144

*All elementary functions except square root require a subroutine called POLY, which takes 42 locations.

APPENDIX H

FORTRAN REFERENCE INFORMATION

Table H-1. FORTRAN Statement Types

Statement	Executable	Non-Executable
ARITHMETIC		
ASSIGNMENT	X	
BACKSPACE	X	
CALL	X	
COMMON		X
CONTINUE	X	
DIMENSION		X
DO	X	
DOUBLE PRECISION		X
END		X
ENDFILE	X	
EQUIVALENCE		X
FORMAT		X
FUNCTION		X
GOTO	X	
IF	X	
PAUSE	X	
READ	X	
RETURN	X	
REWIND	X	
STOP	X	
SUBROUTINE		X
WRITE	X	

Table H-2. FORTRAN I/O Unit Assignments

The following logical unit numbers are associated with the indicated devices at execution time.

Logical Unit 0:	Teletype keyboard and printer
Logical Unit 1:	Teletype paper tape reader and punch
Logical Unit 2:	High speed paper tape reader/punch
Logical Unit 3:	Card reader/punch
Logical Unit 4:	Line printer
Logical Unit 8:	Magnetic tape unit 0
Logical Unit 9:	Magnetic tape unit 1
Logical Unit 10:	Magnetic tape unit 2
Logical Unit 11:	Magnetic tape unit 3

H-3. FORTRAN Memory Maps

H-3 (a) Compile Memory Map (8K)

20

17

16

15

14

13

12

11

10

7

6

5

4

3

2

1

0.7

0.6

0.5

0.4

0.3

0.2

0.1

0.0

Binary Load/Dump	
Unused	
AID II	
Compiler Data Pool	
Compiler Processors	
Compiler Input/Output	
Compiler Data Area	
Unused	
Entry and Interrupt Routines	

H-3. FORTRAN Memory Maps (Continued)

H-3 (b) Load Time Memory Map (8K)

20

17

16

15

14

13

12

11

10

7

6

5

4

3

2

1

0.7

0.6

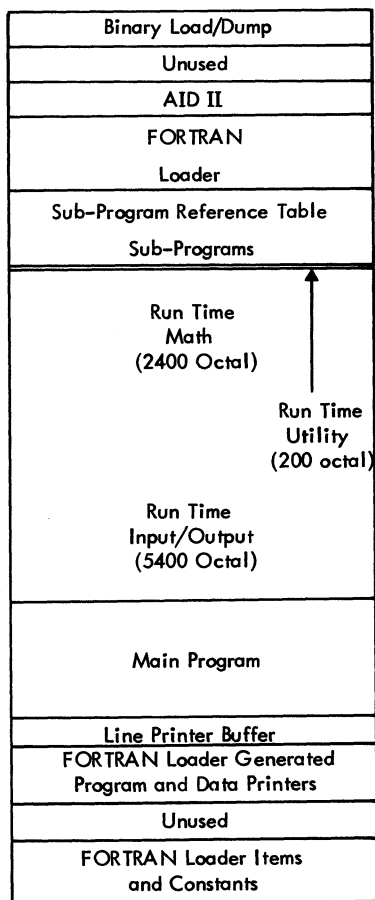
0.5

0.4

0.2

0.1

0.0



H-3. FORTRAN Memory Maps (Continued)

H-3 (c) Execution Time Memory Map (8K)

20

17

16

15

14

13

12

11

10

7

6

5

4

3

2

1

0.7

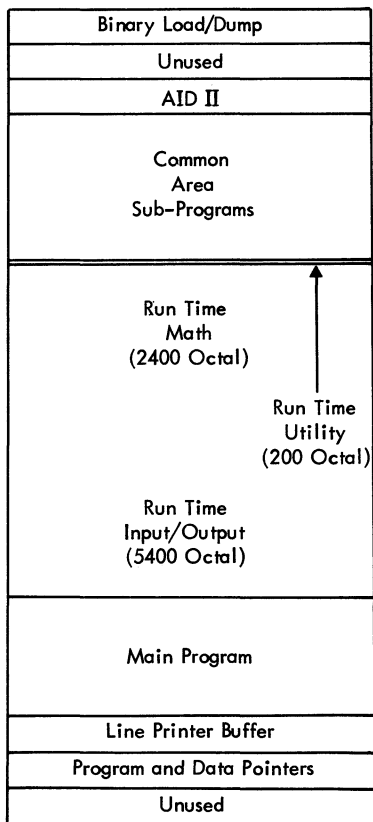
0.6

0.5

0.4

0.2

0.0



Appendix H-4. FORTRAN Object Record Format

General

Each FORTRAN generated program will consist of a series of records, the first of which is marked as the first record of the program. All programs are terminated by an end of program word, and for main programs, an end of tape record. If a program is a function or a subroutine, the first data field of the first record will contain the subprogram name and entry address.

Record Structure

FORTRAN object records are a fixed length of 64 words. Word 1 is unused. Word 2 is the record control word. Words 3 through 5 contain the program name. Words 6 through 63 contain data fields. Word 64 contains the checksum, which is the exclusive OR of words 1 through 63.

Record Control Word Format

BIT 0:	Checksum is present
BIT 1:	End of tape
BIT 2:	End of program
BIT 3:	Start of program
BIT 4:	FORTRAN main program
BIT 5:	FORTRAN subprogram
BIT 6:	Machine language subprogram.

Program Name Format

Six 6-bit characters in packed FORTRAN format. High order starting at bit 3 of word 3, low order ending at bit 0 of word 5. (Bits 16 and 17 unused.)

Data Field Format

Data fields are either two or four word entries. Two word entries consist of a control word and a data word. Four word entries

consist of a control word, two name words and a data word.

Control Word Format

Code	Subcode	Pointer	Name
15 14 13	12 11 10 9 8	7 6 5 4	3 2 1 0

Code Values

- (0) Refer to subcode for specific action.
- (1) Add the location of the selected pointer to the data word (2) before loading it, unless pointer 1 is specified, in which case lower the location by bits 0 through 8 of the data word.
- (2) Add the value of the selected pointer to the data word (2) before loading it.
- (4) Load the data word (2) absolute.

Subcode Values

- (0) Ignore this entry (1 word only).
- (1) Set the loading location counter to the value of the selected pointer plus the data word (2).
- (2) Chain the current loading location counter value to the chain whose last location is indicated by the selected pointer and the data word (2). Stop chaining when an absolute zero address is encountered.
- (6) Terminated error at compile time. Discontinue loading.

- (7) Program generated successfully.
- (10) Define subprogram with name and entry point given in the data word (4).
- (11) Define a region for the pointer indicated whose size is given in the data word (4). Name is given for labeled common regions.
- (12) Call an external subroutine with the name given. The chain address is given by the selected pointer and the data word (4).

Pointer Values

- (0) Program and embedded data region.
- (1) Non-common, non-embedded data region.
- (2) Blank common region.
- (3-31) Labeled common region (not currently implemented).

Name

The first four bits of the first character of a five-character name.

Name Format

Names are five six-bit characters starting in bit 3 of the control word and ending with bit 0 of the second name word.

Data Words

Data words contain instructions, constants, chain addresses, entry addresses, and address offset values.

Paper Tape Format

Paper tape object programs are preceded and followed by 6.4 inches of channel 8 leader. Paper tape records are preceded by a visual record mark (3 frames of rubouts, 377 octal) and a binary record mark (1 zero frame). Each word of the record is punched in three frames of paper tape, 6 bits per frame, high order first. For each frame channel 8 is not punched, channel 7 is the logical complement of channel 6, and bits 6 through 1 are two octal digits of the word.

Channel		8	7	6	5	4	3	2	1
3 Frames (1 word)	{			17	16	15	14	13	12
		*	**	11	10	9	8	7	6
				5	4	3	2	1	0

*Blank channel

**Complement of channel 6

Table H-5. Basic Intrinsic Functions for FORTRAN

Intrinsic Function	Definition	Number of Arguments	Symbolic Name	Type of Argument	Type of Function
Absolute Value	$ a $	1	ABS IABS DABS	Real Integer Double	Real Integer Double
Truncation	Sign of a times largest integer $\leq a $	1	AINT INT IDINT	Real Real Double	Real Integer Integer
Remaindering*	$a_1 \text{ (mod } a_2)$	2	AMOD MOD	Real Integer	Real Integer
Choosing Largest Value	$\text{Max}(a_1, a_2, \dots)$	≥ 2	AMAXO AMAXI MAXO MAXI DMAXI	Integer Real Integer Real Double	Real Real Integer Integer Double
Choosing Smallest Value	$\text{Min}(a_1, a_2, \dots)$	≥ 2	AMINO AMINI MINO MINI DMINI	Integer Real Integer Real Double	Real Real Integer Integer Double
Float	Conversion from integer to real	1	FLOAT	Integer	Real
Fix	Conversion from real to integer	1	IFIX	Real	Integer
Transfer of Sign	Sign of a_2 times $ a_1 $	2	SIGN ISIGN DSIGN	Real Integer Double	Real Integer Double
Positive Difference	$a_1 - \text{Min}(a_1, a_2)$	2	DIM IDIM	Real Integer	Real Integer
Obtain Most Significant Part of Double Precision Argument		1	SNGL	Double	Real
Express Single Precision Argument in Double Precision Form		1	DBLE	Real	Double

*The function MOD or AMOD (a_1, a_2) is defined as $a_1 - [a_1/a_2] a_2$, where $[x]$ is the integer whose magnitude does not exceed the magnitude of x and whose sign is the same as x .

Table H-6. Basic External Functions

External Function	Definition	Number of Arguments	Symbolic Name	Type of Argument	Type of Function
Exponential	e^a	1	EXP	Real	Real
		1	DEXP	Double	Double
Natural Logarithm	$\log_e (a)$	1	ALOG	Real	Real
		1	DLOG	Double	Double
Common Logarithm	$\log_{10} (a)$	1	ALOG10	Real	Real
			DLOG10	Double	Double
Trigonometric Sine	$\sin (a)$	1	SIN	Real	Real
			DSIN	Double	Double
Trigonometric Cosine	$\cos (a)$	1	COS	Real	Real
			DCOS	Double	Double
Hyperbolic Tangent	$\tanh (a)$	1	TANH	Real	Real
Square Root	$(a)^{1/2}$	1	SQRT	Real	Real
			DSQRT	Double	Double
Arctangent	$\arctan (a)$	1	ATAN	Real	Real
		1	DATAN	Double	Double
	$\arctan (a_1/a_2)$	2	ATAN2	Real	Real
		2	DATAN2	Double	Double
Remaindering*	$a_1 \pmod{a_2}$	2	DMOD	Double	Double

*The function DMOD (a_1, a_2) is defined as $a_1 - [a_1/a_2] a_2$, where $[x]$ is the integer whose magnitude does not exceed the magnitude of x and whose sign is the same as the sign of x .

Appendix H-7. Glossary of FORTRAN Terms

actual argument

an argument contained in a function reference or CALL statement

alphanumeric character

an alphabetic or numeric character

argument

a parameter used to pass data between programs and procedures

arithmetic expression

a sequence of constant, variable, or function references connected by arithmetic operators

arithmetic operator

one of the following characters with its associated connotation:

- + (addition)
- (subtraction)
- * (multiplication)
- / (division)
- ** (exponentiation)

array

an ordered set of data of one or two dimensions

array element

one of the members of the set of data of an array

array name

a name that is defined in a DIMENSION statement

column

a character position in a line

comment line

a line with the character C in column 1

continuation line

a line that contains any character other than the digit zero or the character blank in column 6 and that contains blank characters in columns 1 through 5. A continuation line may only follow an initial line or another continuation line.

constant

a name that references a value. A constant may not be redefined

data type

the type of data, either integer or real

dummy

a dummy argument

dummy argument

an argument used to indicate data type, number, and order of procedure arguments.

end line

a program unit terminator

executable program

a main program with possible one or more subprograms

executable statement

a statement that specifies an action of the program. An arithmetic assignment statement, control statement, or input-output statement

expression

an arithmetic expression

external procedure

a subprogram external to a program unit

FORTRAN character set

all alphanumeric and special characters listed on pages 1-1 and 1-2

function

a function subprogram, intrinsic function, or statement function

function reference

a function name followed by an actual argument list contained in parentheses

function subprogram

a FUNCTION statement followed by program body

initial line

a line that is neither a comment line nor an end line and that contains the digit zero or the character blank in column 6

integer

a datum which assumes only integral values. It may assume positive, negative, and zero values.

integer constant

a constant that references an integer value

integer variable

an integer datum that is identified by a symbolic name beginning with any one of the characters I, J, K, L, M, or N

line

a string of 72 characters each of which is a valid FORTRAN character. The character positions in a line are called columns and are consecutively numbered from left to right beginning with column 1.

list

a set of identifiable elements, each of which is separated from its successor by a comma

main program

a program body

name

an element of a statement which is used to reference objects such as data or procedures

non-executable statement

a statement that describes the characteristic and arrangement of data, editing information, statement functions, and classification of program units

operator

an element of a statement which specifies an action upon named objects

procedure

a function or subroutine

processor

the program which processes FORTRAN programs

program

a collection of statements, comment lines, and end lines

program body

a collection of optional specification statements optionally followed by statement function definition, followed by a program part, followed by an end line

program unit

a main program or subprogram

program part

at least one executable statement. A program part may but need not contain FORMAT statements

real

a datum which is a processor approximation to the value of a real number. A real datum assumes both integral and fractional values and may assume positive, negative, and zero values

real constant

a constant that references a real value

real variable

a datum that is identified by a symbolic name beginning with any character other than I, J, K, L, M, or N

reference

a verb indicating an identification of a datum and implying that the current value of the datum will be made available

signed constant

a constant preceded by a plus or minus sign

special character

one of the ten characters: blank, equals, plus, minus, asterisk, slash, left parenthesis, right parenthesis, comma, and decimal point

specification statement

a COMMON, DIMENSION, or EQUIVALENCE statement

statement

an initial line optionally followed by up to five ordered continuation lines. The statement is contained in columns 7 through 72 of the lines

statement label

one of four digits, the value of which must be greater than zero. Leading zeros are not significant

string

a series of data

subprogram

a SUBROUTINE or FUNCTION statement followed by a program body containing at least one RETURN statement

subroutine

a subroutine subprogram

subroutine subprogram

a SUBROUTINE statement followed by a program body

subscript

a parenthesized list of subscript expressions

subscript expressions

any one of the following expressions: $C*V+K$, $C*V-K$, $C*V$, $V+K$, $V-K$, V , K , where C and K are integer constants and V is an integer variable reference

symbolic name

one of five alphanumeric characters, the first of which must be alphabetic

variable

a datum that is identified by a symbolic name

SALES OFFICES

United States

CALIFORNIA

216 Pico Boulevard
Santa Monica, California 90405
(213) 392-4871
TWX: 910-343-6965

4940 El Camino Real
Los Altos, California 94022
(415) 968-9996
TWX: 910-379-6446

CONNECTICUT

Vernon Professional Center
Route #30
Vernon, Connecticut 06086
(203) 647-1500
TWX: 710-427-7609

ILLINOIS

205 West Touhy Avenue
Park Ridge, Illinois 60068
(312) 692-7184
TWX: 910-253-1824

TEXAS

3939 Hillcroft Ave.
Suite 180
Houston, Texas 77036
(713) 781-0105
TWX: 910-881-2620

PENNSYLVANIA

520 Penn Avenue
P.O. Box 291
Fort Washington, Pa. 19034
(215) 643-2355

WASHINGTON, D.C.

11215 Oak Leaf Drive
Silver Springs, Maryland 20921
(301) 593-8750

Foreign

AUSTRALIA

Varian Pty. Ltd.
38 Oxley Street
Crows Nest
Sydney, Australia
Tel: 43-0673

FRANCE

Varian S.A.
85 rue Fondary
Paris XV, France
Tel: 306-9811
TELEX: 27642

GERMANY

Varian GmbH
Breitwiesenstrasse 9
7 Stuttgart-Vaihingen
Germany
Tel: 78 33 51
TELEX: 72 31 24

SWEDEN

Varian A.B.
Skyttenholmsvägen 7d
Solna, Sweden
Tel: 82 00 30
TELEX: 10 403

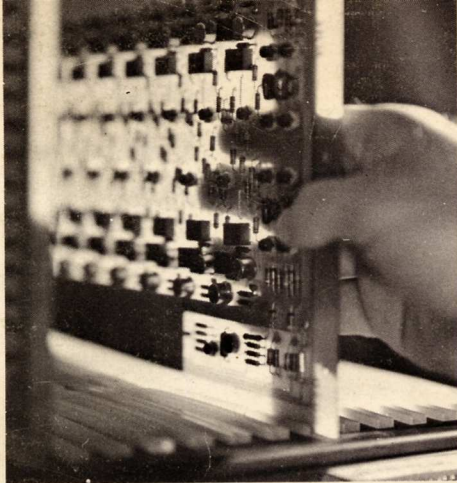
SWITZERLAND

Varian A.G.
Baarerstrasse 77
6300 Zug, Switzerland
Tel: (042) 4 45 55
TELEX: 58 444

UNITED KINGDOM & IRELAND

Varian Associates, Ltd.
Russell House
Molesey Road
Walton-on-Thames
Surrey, England
Tel: Walton 28 766
TELEX: 26 13 51

\$3.00



varian data 620/i computer manual



varian data machines

2722 Michelson Drive
Irvine, California 92664

